Encl (1)

12

AD A138935

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) IMPLEMENTATION OF AN OPTIMAL MULTICOMMODITY NETWORK FLOW ALGORITHM BASED ON GRADIENT PROJECTION AND A PATH FLOW FORMULATION		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER LIDS-P-1364
7. AUTHOR(s) Dimitri P. Bertsekas, Bob Gendron, Wei K. Tsai		8. CONTRACT OR GRANT NUMBER(s) ARPA Order No. 3045/5-7-75 ONR/N00014-75-C-1183
9. PERFORMING ORGANIZATION NAME AND ADDRESS Massachusetts Institute of Technology Laboratory for Information and Decision Systems Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Code No. 5T10 ONR Identifying No. 049-383
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE February 1984
		13. NUMBER OF PAGES 64
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited

DTIC ELECTE MAR 8 1984 S A D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DTIC FILE COPY

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The implementation of a multicommodity flow algorithm into a FORTRAN code is discussed. The algorithm is based on a gradient projection method [1] with diagonal scaling based on Hessian or Jacobian information. The flows carried by the active paths of each origin-destination (OD) pair are iterated upon one OD pair, per iteration. The data structures and memory requirements of the algorithm are discussed and are compared with those of other formulations based on link flows associated with each origin, and aggregate link flows.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

84 03 08 005

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

February 1984 LIDS-P-1364

IMPLEMENTATION OF AN OPTIMAL MULTICOMMODITY NETWORK FLOW ALGORITHM BASED ON GRADIENT PROJECTION AND A PATH FLOW FORMULATION†

by

Dimitri P. Bertsekas, Bob Gendron and Wei K. Tsai*

ABSTRACT

The implementation of a multicommodity flow algorithm into a FORTRAN code is discussed. The algorithm is based on a gradient projection method [1] with diagonal scaling based on Hessian or Jacobian information. The flows carried by the active paths of each origin-destination (OD) pair are iterated upon one OD pair at a time. Active paths are generated using a shortest path algorithm--one path per OD pair, per iteration. The data structures and memory requirements of the algorithm are discussed and are compared with those of other formulations based on link flows associated with each origin, and aggregate link flows.

†This research was supported by the National Science Foundation under grant NSF-ECS-8217668 and the Defense Advanced Research Projects Agency under grant ONR/N00014-75-C-1183.

*D. P. Bertsekas and W. K. Tsai are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139. B. Gendron is with Alphatech, Inc., Burlington, MA.

1. Optimal Multicommodity Flow Problem Formulation

We have a directed network with set of nodes L and set of links N. Let W be a collection of ordered node pairs referred to as origin-destination (OD) pairs. For each OD pair $w \in W$ we are given a positive number r_w representing input flow into the network from origin to destination. Let P_w be a given set of directed paths joining the origin node and destination node of OD pair w. (P_w could be the set of all simple directed paths joining origin and destination, or it could be a restricted set of paths determined a priori on the basis of some unspecified considerations). Note that we do not exclude the possibility that two distinct OD pairs have the same origin and destination and possibly a different set of paths, but are associated with different classes or types of traffic.

Let x_p be the flow carried by a generic path p. The optimization variables of the problem are x_p, $p \in P_w$, $w \in W$ and must satisfy the constraints

$$\sum_{p \in P_w} x_p = r_w \quad , \quad \forall\, w \in W, \tag{1}$$

$$x_p \geq 0 \quad , \quad \forall\, p \in P_w,\ w \in W. \tag{2}$$

Let x be the vector of all path flows

$$x = \{x_p \mid p \in P_w,\ w \in W\} \tag{3}$$

For each link (i,j) and OD pair w we are given a continuously differentiable function $T_{ij}(x,w)$, which is to be interpreted as the length of link (i,j) when the path flow vector is x. In data communication routing and traffic assignment problems $T_{ij}(x,w)$ usually has the interpretation of marginal delay and travel time respectively (see [1]-[19]). We assume that for all feasible x and all $w \in W$

$$T_{ij}(x,w) \geq 0 \quad , \quad \forall\ (i,j) \in L \ , \tag{4}$$

The length of a path $p \in P_w$ when the path flow vector is x is defined by

$$L_p(x,w) = \sum_{(i,j)\in p} T_{ij}(x,w) \tag{5}$$

i.e. it is the sum of lengths of its links.

The problem we are considering is the following:

Find a path flow vector x* satisfying the constraints (1), (2) and such that for every $w \in W$ and $p \in P_w$

$$x_p^* > 0 \implies L_p(x^*,w) \leq L_{p'}(x^*,w), \quad \forall\ p' \in P_w. \tag{6}$$

In other words we are looking for a path flow pattern x* whereby the only paths that carry positive flow are shortest paths with respect to the link lengths $T_{ij}(x^*,w)$.

The problem described above includes, among others, problems of optimal routing in data networks [1]-[8] and (possibly asymmetric) traffic assignment problems in transportation networks [9]-[19]. We refer to the references just cited for extensive discussions. The survey paper [1] describes in detail the data communication context. A typical formulation there is to find a feasible path flow vector x that minimizes

$$\sum_{(i,j)} D_{ij}(F_{ij}) \tag{7}$$

where D_{ij} is a monotonically increasing, twice differentiable function of the total flow F_{ij} of the link (i,j) given by

$$F_{ij} = \sum_{w \in W} \sum_{p \in P_w} x_p \, \delta(p,i,j) \tag{8}$$

where

$$\delta(p,i,j) = \begin{cases} 1 & \text{if link } (i,j) \text{ belong to path } p \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

It can be shown (see e.g. [1]) that if we make the identification

$$T_{ij} = D'_{ij} : \text{ The first derivative of } D_{ij} \tag{10}$$

the routing optimization problem falls within the framework of the general multicommodity flow problem described earlier.

2. A Projection Method for Solving the Multicommodity Flow Problem

The MULTIFLO and MULTIFLO1 codes given in Appendices I and II of this report implement an algorithm that solves the problem of the previous section for the case where for all OD pairs $w \epsilon W$

P_w = Set of all simple paths joining the origin and destination of w.

The set of OD pairs is divided into C groups called commodities. All OD pairs of a commodity have the same origin node. Furthermore the data structures of the codes can handle only the case where the lengths $T_{ij}(x,w)$ depend on w through the corresponding commodity c. That is

$$T_{ij}(x,w) = T_{ij}(x,\bar{w}), \ \forall\ (i,j)\epsilon L, \text{ and OD pairs } w,\ \bar{w} \text{ of the same commodity } c.$$

It is also assumed that for all feasible F

$$\frac{\partial T_{ij}}{\partial x_p} \geq 0 \qquad \forall\ (i,j) \quad \text{belonging to the path } p$$

MULTIFLO and MULTIFLO1 operate as follows:

At the beginning of the kth iteration we have for the generic OD pair w W a set of active paths P_w^k consisting of at most (k-1) paths. (These paths were generated in earlier iterations and it is implicitly assumed that all other paths carry zero flow). The following calculation is executed sequentially for each commodity--first for commodity 1, then for commodity 2, and so on up to the last commodity C:

<u>Step 1</u>: A shortest path that joins the origin node for the commodity with all other nodes is calculated. The length for each link (i,j) used for this calculation is $T_{ij}(x,w)$ where x is the current path flow vector. These shortest paths are added to the corresponding list of active paths of each OD pair of the commodity if they are not already there, so now the list of active paths for each OD pair of the commodity contains at most k paths.

<u>Step 2</u>: Each OD pair w of the commodity is taken up sequentially. For each active path p of w the length L_p [cf. (5)] is calculated together with an additional number α_p called the <u>stepsize</u> (more on the choice of this later). Both L_p and α_p are calculated on the basis of the current total link flow vector. Let $\bar{p}$ be the shortest path calculated in Step 1 for the OD pair. The path flows of all paths $p \neq \bar{p}$ are updated according to

$$x_p \leftarrow \begin{cases} \max\{0,\ x_p-\alpha_p(L_p-L_{\bar{p}})\} & \text{if } L_p > L_{\bar{p}} \\ x_p & \text{otherwise.} \end{cases} \tag{11}$$

The path flow of the shortest path $\bar{p}$ is then adjusted so that the sum of flows of all active paths equals r_w as required by the constraint (1), i.e.

$$x_{\bar{p}} \leftarrow r_w - \sum_{\text{active } p\neq\bar{p}} x_p. \tag{12}$$

In other words an amount x_p or $\alpha_p(L_p-L_{\bar{p}})$ is shifted from each nonshortest path to the shortest path $\bar{p}$--whichever is smaller. The total link flows F_{ij} are adjusted to reflect the changes in x_p and $x_{\bar{p}}$.

The rationale for iteration (11) is explained in [1], [6], [8], [9].

It is based on a gradient projection method [9], [21]. Note that it is possible that $L_p < L_{\bar{p}}$ for some $p \neq \bar{p}$ even though $\bar{p}$ was calculated earlier as a shortest path. The reason is that by the time L_p and $L_{\bar{p}}$ are computed the total link flow vector may have changed since the time the shortest path has been calculated due to iterations on the path flows of other OD pairs of the same commodity.

Regarding the choice of the stepsize α_p, the MULTIFLO and MULTIFLO1 codes use the following formula for all $p \neq \bar{p}$

$$\alpha_p = S_p^{-1} \tag{13}$$

where

$$S_p = \sum_{(i,j)\in L_p} \frac{\partial T_{ij}}{\partial x_p} \tag{14}$$

and L_p is the set of links

$$L_p = \{(i,j) \mid (i,j) \text{ belongs to either } p \text{ or } \bar{p}, \text{ but not to both } p \text{ and } \bar{p}\}.$$

The rationale for this is as follows:

If we interpret the algorithm as one that tries to satisfy the equation

$$L_p - L_{\bar{p}} = 0, \qquad \forall\, p \text{ with } x_p > 0, \tag{16}$$

a natural choice for α_p is

$$\hat{\alpha}_p = \frac{\Delta x_p}{\Delta(L_p - L_{\bar{p}})} \tag{17}$$

where $\Delta(L_p-L_{\bar{p}})$ is the variation of $(L_p-L_{\bar{p}})$ resulting from a small variation

Δx_p in the path flow x_p (and an attendant variation $-\Delta x_p$ in the path flow $x_{\bar{p}}$). This corresponds to an approximate form of Newton's method whereby only the diagonal elements of the Jacobian matrix (corresponding to the current OD pair) are taken into account while the off-diagonal terms are set to zero (see also [1] for further discussion). For $\Delta x_p \to 0$ it is easily seen that (17) yields

$$\hat{\alpha}_p^{-1} = \sum_{(i,j)\varepsilon p} \left(\frac{\partial T_{ij}}{\partial x_p} - \frac{\partial T_{ij}}{\partial x_{\bar{p}}}\right) + \sum_{(i,j)\varepsilon \bar{p}} \left(\frac{\partial T_{ij}}{\partial x_{\bar{p}}} - \frac{\partial T_{ij}}{\partial x_p}\right). \tag{18}$$

In most cases of interest we have

$$\frac{\partial T_{ij}}{\partial x_p} \simeq \frac{\partial T_{ij}}{\partial \bar{x}_p} \qquad \text{if } (i,j)\varepsilon p \text{ and } (i,j)\varepsilon \bar{p}$$

$$\frac{\partial T_{ij}}{\partial x_p} \simeq 0 \qquad \text{if } (i,j)\not\varepsilon p$$

$$\frac{\partial T_{ij}}{\partial \bar{x}_p} \simeq 0 \qquad \text{if } (i,j)\not\varepsilon \bar{p}$$

so (18) becomes approximately [c.f. (18), (14)]

$$\hat{\alpha}_p^{-1} \simeq \sum_{(i,j)\varepsilon L_p} \frac{\partial T_{ij}}{\partial x_p} = S_p,$$

thereby justifying the use of the stepsize (13), (14).

If one wishes to employ the formula (18) for the stepsize it is necessary to modify the codes. These modifications should not be too difficult for an experienced user. Another possibility is to use a smaller value of stepsize α_p than the one given by (13)--for example $\alpha_p = \rho S_p^{-1}$ $\rho\varepsilon(0,1)$ is a fixed relaxation parameter. (A smaller stepsize enhances the convergence properties of the algorithm but may deteriorate its rate of convergence). This can be accomplished without any changes in the code by simply introducing the relaxation parameter ρ in the subroutine that calculates $\frac{\partial T_{ij}}{\partial x_p}$ [cf. (14)].

In the MULTIFLO code a shortest path tree is generated and stored at each iteration for each commodity. As a result the memory storage for shortest paths is proportional to the number of iterations so for large problems one cannot execute a large number of iterations without incurring a heavy penalty for disk I/O. MULTIFLO will usually find in five to ten iterations what is for most practical problems an adequate approximation to an optimal solution. This is particularly true of lightly loaded networks (e.g. with utilization of all links less than 60% at the optimum). For heavily loaded networks the number of required iterations usually tends to be larger (say 10-30). It should be a rare occasion when a user will require more than thirty iterations for his practical problem.

MULTIFLO1 differs from MULTIFLO only in the method used for storing the active paths. MULTIFLO1 stores explicitly all active paths in a single array rather than storing them implicitly through the generated shortest path trees. As a result the memory storage of MULTIFLO1 depends on the number of active paths generated and is largely independent of the number of iterations executed. For certain problems including situations where a large number of iterations is desired MULTIFLO1 may hold a storage advantage over MULTIFLO. Both codes generate identical numerical results although MULTIFLO1 appears to be somewhat faster on sample test problems.

3. Data Structures for Representing the Problem

The data structures of MULTIFLO and MULTIFLO1 are described in the code documentation. The problem input structure will be illustrated here by means of the 5 node-6 link network shown in Figure 1:

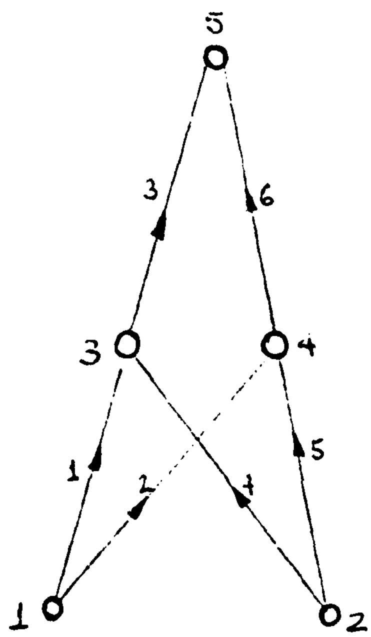

Figure 1

Node Length Arrays (FRSTOU, LASTOU):

These arrays specify the network topology.

FRSTOU(NODE): The first link out of NODE

LASTOU(NODE): The last link out of NODE

NODE	FRSTOU	LASTOU
1	1	2
2	4	5
3	3	3
4	6	6
5	0	0

Note that all arcs with the same head node must be grouped together in the arc list. A node with no outgoing links is recognized via FRSTOU = 0

Arc Length Arrays (STARTNODE, ENDNODE)

These arrays also specify the network topology:

STARTNODE (ARC): The head node of ARC

ENDNODE (ARC): The tail node of ARC

ARC	STARTNODE	ENDNODE
1	1	3
2	1	4
3	3	5
4	2	3
5	2	4
6	4	5

Commodity Length Arrays (ORGID, STARTOD)

ORGID (COMMODITY): The origin node of COMMODITY

STARTOD (COMMODITY): A pointer to the first OD pair of COMMODITY on the OD pair list

For the example of Figure 1 we will assume three commodities

COMMODITY	ORGID	STARTOD
1	2	1
2	1	3
3	1	4

Note that it is required that OD pairs are listed sequentially by commodity, i.e. the OD pairs of commodity 1 are listed first, followed by the OD pairs of commodity 2, etc. Therefore the STARTOD array together with the total number of OD pairs specify all OD pairs associated with each commodity.

OD Pair Length Arrays (DEST, INPUT_FLOW)

DEST(OD): The destination node of OD

INPUT_FLOW(OD): The input traffic of OD

OD	DEST	INPUT_FLOW
1	3	problem dependent
2	5	"
3	3	"
4	4	"
5	5	"

From the arrays ORGID, STARTOD and DEST together with the total number of OD pairs the set of OD pairs corresponding to each commodity is completely specified. For our example these are:

COMMODITY	OD PAIRS
1	(2,3), (2,5)
2	(1,3)
3	(1,4), (1,5)

Additional input information is required to calculate the link lengths T_{ij} and their first derivatives $\frac{\partial T_{ij}}{\partial x_p}$ in the subroutine DERIVS and DERIV1. This is of course problem dependent. The listing of Appendix I gives an example which is typical of routing problems in data networks [cf. equations (7)-(10)].

4. Memory Requirements - Comparisons with Other Methods

The memory storage requirements of both MULTIFLO and MULTIFLO1 are substantial, but this is true for all methods that provide as output not only the optimal total link flows but also detailed information about the optimal routing from origins to destinations (i.e. optimal path flows).

Assuming that 1 byte is allocated for a logical variable, 2 bytes are allocated for storing a node or link identification number and an iteration number, 4 bytes are allocated for storing a commodity, OD pair or path identification number, and 4 bytes are allocated for storing a real number (e.g. a path or link flow) the total array storage in bytes of MULTIFLO during execution is

$$6n_N + 9n_L + 6n_C + 6n_{OD} + 10n_P + 2n_I\, n_N\, n_C \tag{19}$$

where:

n_N: Number of nodes

n_L: Number of links

n_C: Number of commodities

n_{OD}: Number of OD pairs

n_P: Number of active paths generated

n_I: Number of iterations.

Additional storage is required for information necessary to calculate link lengths and their derivatives but this is typically of order $O(n_L)$ and is not significant.

The dominant array as far as storage of MULTIFLO is concerned is the triple indexed PRED array which stores the shortest path trees generated for each commodity at each iteration. This array accounts for the last term $2n_I n_N n_C$ in (19). The term $10n_P$ is also substantial since the number of active paths n_P can be as large as $n_I n_{OD}$. However, because the algorithm stores a path only once at the iteration it is first generated and does not duplicate it if it is generated again later, the actual number n_P is typically much smaller than $n_I n_{OD}$. This was confirmed by extensive computational experimentation, that showed that except for very heavily loaded networks the actual number of active paths n_P was typically no more than $2n_{OD}$(!) and often considerably less. We conclude therefore that the dominant bottleneck for storage is the shortest path description array PRED requiring $2n_I n_N n_C$ bytes.

In the MULTIFLO1 code the array PRED is not used. In its place the array PDESCR is used which requires storage of $2n_P n_N$ at most. This calculation assumes conservatively that a path has n_N links. However in practice the actual storage for PDESCR is several times less than $2n_P n_N$. If we adopt the rough estimate $n_P \simeq 2n_{OD}$ then we conclude that the storage requirements of MULTIFLO and MULTIFLO1 are roughly comparable if the number of iterations n_I is comparable to something between $\frac{n_{OD}}{n_C}$ and $\frac{n_{OD}}{4n_C}$ with MULTIFLO1 becoming definitely preferable if $n_I \simeq \frac{n_{OD}}{n_C}$. MULTIFLO1 is also preferable for problems that are solved repetitively with minor variations in their data since then the knowledge of the path description array PDESCR can be fruitfully exploited. This is not possible with MULTIFLO.

In large problems where only the total link flows are of interest (e.g. traffic assignment problems) a different algorithm [e.g. the flow

Deviation (or the Frank-Wolfe) method [3], [8] or the Cantor-Gerla (or simplicial approximation) method [4], [15], may be preferable over MULTIFLO or MULTIFLO1, since then storage of order $O(n_L)$ or perhaps $O(n_I n_L)$ is required. However when detailed routing information is of interest the memory storage requirements of MULTIFLO are competitive with those of other methods based on shortest paths including the Flow Deviation and Cantor-Gerla methods. The reason is that detailed routing information can be provided by these methods only if the shortest paths generated at each iteration are stored explicitly in an array such as PRED, and as mentioned earlier this is the main memory storage bottleneck.

There are algorithms that can solve multicommodity flow problems and provide detailed routing information without requiring the generation and storage of shortest paths. These algorithms are based on a link flow formulation [20], or the link flow fraction formulation due to Gallager [2], [5], [7] whereby the optimization variables are the flows or fractions of flow respectively for each commodity that are routed along each link. The storage requirement for these algorithms is of order $O(n_C n_L)$ and is independent of the number of iterations. When we compare this storage with the $O(n_I n_C n_L)$ storage of algorithms based on shortest paths we see that link flow formulations hold an advantage in terms of storage for problems where a large number of iterations is desirable. The reverse is true if the number of iterations required for adequate solution of the problem is small, or if the number of links is much larger than the number of nodes.

We finally note a final advantage of the path flow formulation over link flow formulations. When the set of paths for each OD pair is restricted to be a given strict subset of the set of all possible simple paths it is extremely cumbersome to use a link flow formulation. By contrast it is straightforward to modify the MULTIFLO1 code to handle this situation.

References

[1] Bertsekas, D.P., "Optimal Routing and Flow Control Methods for Communication Networks", in Analysis and Optimization of Systems (Proc. of 5th International Conference on Analysis and Optimization of Systems, A. Bensoussan and J. L. Lions, eds., Versailles, France), Springer-Verlag, Berlin and N.Y., 1982, pp. 615-643.

[2] Gallager, R.G., "A Minimum Delay Routing Algorithm Using Distributed Computation", IEEE Trans. on Communications, Vol. COM-25, 1978, pp. 73-85.

[3] Fratta, L., M. Gerla, and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design", Networks, Vol. 3, 1973, pp. 97-133.

[4] Cantor, D.G., and M. Gerla, "Optimal Routing in a Packet Switched Computer Network", IEEE Trans. on Computers, Vol. C-23, 1974, pp. 1062-1069.

[5] Bertsekas, "Algorithms for Nonlinear Multicommodity Network Flow Problems", International Symposium on Systems Optimization and Analysis, A. Bensoussan and J. L. Lions (eds.), Springer-Verlag, 1979, pp. 210-224.

[6] Bertsekas, D.P., "A Class of Optimal Routing Algorithms for Communication Networks", Proc. of 5th International Conference on Computer Communication (ICCC-80), Atlanta, Ga., Oct. 1980, pp. 71-76.

[7] Bertsekas, D.P., E.M. Gafni, and R.G. Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks", LIDS Report LIDS-P-1082, M.I.T., March 1981.

[8] Bertsekas, D.P. and E.M. Gafni, "Projected Newton Methods and Optimization of Multicommodity Flows", IEEE Trans. on Aut. Control, Dec. 1983.

[9] Bertsekas, D.P. and E.M. Gafni, "Projection Methods for Variational In-Equalities with Application to the Traffic Assignment Problem", Math. Prog. Study, 17, D. C. Sorensen and R. J.-B. Wets (eds.), North-Holland Amsterdam, 1982, pp. 139-159

[10] Aashtiani and T.L. Magnanti, "Equilibria on a Conjested Transportation Network", SIAM J. of Algebraic and Discrete Math., Vol. 2, 1981, pp. 213-226.

[11] Florian, M., "An Improved Linear Approximation Algorithm for the Network Equilibrium (Packet Switching) Problem", Proc. of 1977 IEEE Conf. on Dec. and Control, New Orleans, La., 1977, pp. 812-818.

[12] Florian, M. and S. Nguyen, "A Method for Computing Network Equilibrium with Elastic Demand", Trans. Sci., Vol. 8, 1974, pp. 321-332.

[13] Florian, M., "The Convergence of Diagonalization Algorithms for Fixed Demand Asymmetric Network Equilibrium Problems", Centre de Recherche sur les Transports Publ. #198, Jan. 1981.

[14] Nguyen, S., "An Algorithm for the Traffic Assignment Problem", Transportation Science, Vol. 8, 1974, pp. 203-216.

[15] Lawphongpanich, S., and D.W. Hearn, "Simplicial Decomposition of the Asymmetric Traffic Assignment Problem", Univ. of Florida Report, Oct. 1982.

[16] Dafermos, "An Extended Traffic Assignment Model with Applications to Two-Way Traffic", Transportation Science, Vol. 5, 1971, pp. 366-389.

[17] Dafermos, S.C., "Traffic Equilibrium and Variational Inequalities", Transportation Science, Vol. 14, 1980, pp. 42-54.

[18] Aashtiani, H.Z., "The Multi-Model Traffic Assignment Problem", Ph.D. Thesis, Sloan School of Management, M.I.T., Cambridge, Mass. May, 1979.

[19] Pang, J.S. and D. Chan, "Iterative Methods for Variational and Complementarity Problems", Math. Programming, Vol. 24, pp. 284-313.

[20] Dembo, R.S. and J. G. Klincewicz, "A Scaled Reduced Gradient Algorithm for Network Flow Problems with Convex Separable Costs", Math. Programming Study 15, 1981, pp. 125-147.

[21] Bertsekas, D.P., "Projected Newton Methods for Optimization Problems with Simple Constraints", SIAM J. Control and Optimization, Vol. 20, 1982, pp. 221-246.

[22] Gafni, E.M. and D. P. Bertsekas, "Two-Metric Projection Methods for Constrained Optimization", Lab. for Information and Decision Systems Report LIDS-R-1235, M.I.T., Cambridge, Mass., Sept. 1982, SIAM J. on Control & Optimization, to appear.

[23] Pape, U., "Implementation and Efficiency of Moore Algorithms for the Shortest Route Problem", Math. Programming, Vol. 7, 1974, pp. 212-222.

APPENDIX I: MULTIFLO Code

The following FORTRAN code works on the VAX family of computers. It consists of a DRIVER program and several subroutines:

LOAD: Reads network topology and link length data from disk.

MULTIFLO: This is the main algorithm.

SP: Calculates a shortest path tree from an origin node to all other nodes.

PRFLOW: Prints out to disk problem data and algorithmic results.

DERIVS: This user supplied routine calculates for a given link (i,j) its length T_{ij} (D1CAL) and the length derivative $\frac{\partial T_{ij}}{\partial x_p}$ (D2CAL).

DERIV1: This routine is the same as DERIVS except that it calculates the length T_{ij} (D1CAL) but not the length derivative $\frac{\partial T_{ij}}{\partial x_p}$.

DELAY: This user supplied routine is useful only if the multicommodity flow problem is a routing optimization problem of the form (7)-(10) as described in Section 1. For asymmetric traffic assignment problems it has no purpose. It calculates the total delay

$$\sum_{(i,j)} D_{ij}(F_{ij})$$

where $D'_{ij} = T_{ij}$ [cf. (7)-(10)]. The value of $D_{ij}(F_{ij})$ is calculated using the function DCAL.

Two versions of the shortest path routine SP are provided (SHORTPAPE and SHORTHEAP) which can be used interchangeably. SHORTHEAP is recommended for problems where there are only few destinations for each commodity. Otherwise SHORTPAPE based on [23] should be preferable.

A program (SETUP) is also provided for the purpose of creating the data describing the problem in a format that is compatible with the LOAD routine.

The routines LOAD, DERIV1, DERIVS, DELAY, and DCAL supplied in this appendix correspond to the most commonly solved optimal routing problem in data communication network applications whereby a capacity C_{ij} is given for each link (i,j) (this is the array BITRATE in the code) and

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij}-F_{ij}} \quad \text{(M/M/1 Queueing Delay)} \tag{A.1}$$

$$T_{ij}(F_{ij}) = \frac{C_{ij}}{(C_{ij}-F_{ij})^2}$$

$$\frac{\partial T_{ij}(F_{ij})}{\partial F_{ij}} = \frac{2C_{ij}}{(C_{ij}-F_{ij})^3} .$$

Because $D_{ij}(F_{ij}) \to \infty$ as $F_{ij} \to C_{ij}$ these formulas have been modified so that if $F_{ij} \geq \rho\, C_{ij}$, where $\rho \in (0,1)$ is a parameter set by the user, then D_{ij}, T_{ij}, $\frac{\partial T_{ij}}{\partial F_{ij}}$ are calculated using a quadratic function which has the same value, first and second derivatives as $\frac{F_{ij}}{C_{ij}-F_{ij}}$ at the breakpoint ρC_{ij}. In the program the parameter ρ is given by the variable MAXUTI set in the subroutine LOAD to 0.99. The user may wish to change this value. The guideline is that ρ should be set at a value exceeding the maximum link utilization

$$\max_{(i,j)\in L} \frac{F_{ij}}{C_{ij}}$$

at the optimal solution. This trick gets around situations whereby the input flows are so large that exceeding some of the link capacities during some phase of the algorithm is inevitable.

The MULTIFLO code will stop computing when one of two conditions is met: Either the maximum number of iterations (MAXITER) is exceeded or a normalized measure of deviation from the optimal solution falls below a certain tolerance (TOL). This measure is roughly equal to the percentage of input traffic of an OD pair that does not lie on a shortest path (maximized over all OD pairs), and its magnitude is not substantially affected by the size of the problem. Both convergence parameters MAXITER and TOL are set by the user in the subroutine LOAD.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       DRIVER
C
C       'DRIVER' IS A SIMPLE EXECUTIVE TO INVOKE THE 'MULTIFLO' COMMODITY
C       ROUTING PROGRAM.  'DRIVER' INVOKES SUBPROGRAM 'LOAD' TO READ
C       DATA INTO 'MULTIFLO' INPUT COMMON BLOCKS.  FILES READ BY
C       'LOAD' ARE CREATED BY A TERMINAL SESSION WITH THE USER FOR
C       NETWORK DEFINITION THROUGH THE USE OF PROGRAM 'SETUP'.
C
C       EXECUTION STEPS FOR PROGRAM 'DRIVER'
C
C               1) ASSIGN FORTRAN UNIT 01 AS CREATED BY PROGRAM 'LOAD'
C               2) ASSIGN FORTRAN UNIT 02 AS CREATED BY PROGRAM 'LOAD'
C               3) ASSIGN FORTRAN UNIT 06 AS A DESIGNATED OUTPUT FILE
C
C               E.G.:
C                   $ ASSIGN NETWORK.DAT FOR001
C                   $ ASSIGN TRAFFIC.DAT FOR002
C                   $ ASSIGN OUTPUT.DAT  FOR006
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        PROGRAM DRIVER
C
C       LOAD FORTRAN UNIT 01 AND FORTRAN UNIT 02 FROM DISK AS CREATED
C       FROM PROGRAM 'SETUP'
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'PATHS.BLK'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
        INTEGER COMMODITY,ORIGIN,DESTOD,OD,PATH
        CALL LOAD
C
C       EXECUTE THE 'MULTIFLO' NETWORK ALGORITHM.  'MULTIFLO' SCHEDULES
C       ITS OWN OUTPUTS TO FORTRAN UNIT 06 ON EACH ITERATION
C
C       INITIALIZE THE TIMER
        CALL LIB$INIT_TIMER
        CALL MULTIFLO
C       RECORD THE COMPUTATION TIME
        CALL LIB$SHOW_TIMER
C
C          PRINT MAX LINK UTILIZATION (RELEVANT FOR M/M/1 QUEUEING DELAY
C          OPTIMIZATION)
C
          UMAX=0.0
          DO 100 I=1,NA
            UMAX=MAX(UMAX,FA(I)/BITRATE(I))
100       CONTINUE
          WRITE(6,*)'MAXIMUM LINK UTILIZATION'
          WRITE(6,*)UMAX
C
C       PRINT FINAL PATH FLOW INFO
C
        WRITE(6,*)'ORIGIN / DESTINATION / PATH # / PATH_FLOW'
        DO 1000 COMMODITY=1,NUMCOMMOD
          ORIGIN=ORGID(COMMODITY)
          DO 500 OD=STARTOD(COMMODITY),STARTOD(COMMODITY+1)-1
```

```
           DESTOD=DEST(OD)
           PATH=OD
           DO WHILE (PATH.GT.0)
             WRITE(6,*)ORIGIN,DESTOD,PATH,FP(PATH)
             PATH=NEXTPATH(PATH)
           END DO
500      CONTINUE
1000   CONTINUE
       STOP
       END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       LOAD
C
C       'LOAD' READS IN DATA FROM DISK CREATED WITH PROGRAM 'SETUP' FOR
C       USE BY PROGRAM 'MULTIFLO'.  NETWORK SPECIFICATION DATA RESIDES
C       ON FORTRAN UNIT 01 AND NETWORK TRAFFIC SPECIFICATION DATA
C       RESIDES ON FORTRAN UNIT 02.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE LOAD
        IMPLICIT NONE
C
C       **********************  INCLUDE COMMON BLOCKS  ********************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
C
C       ********************  LOCAL VARIABLE DEFINITIONS  *****************
C
        INTEGER I
C               DO LOOP INDEX
C
C       **********************  EXECUTABLE CODE  ***************************
C       TERMINATION PARAMETERS. MAXITER GIVES THE MAX # OF ITERATIONS
C       TOL IS A SOLUTION ACCURACY TOLERANCE. RECOMMENDED VALUES
C       ARE 0.01 TO 0.0001. THE PROPER VALUE OF TOL IS LARGELY
C       INDEPENDENT OF THE PROBLEM SIZE.
        MAXITER=20
        TOL=0.01
C       THE FOLLOWING PARAMETER MAKES SENSE ONLY FOR ROUTING PROBLEMS
C       WHERE AN M/M/1 QUEUING FORMULA IS USED FOR DELAY.
C       IT GIVES THE THRESHOLD FRACTION OF CAPACITY BEYOND WHICH
C       THE DELAY FORMULA IS TAKEN TO BE QUADRATIC.
        MAXUTI=0.99
C
C       LOAD THE NETWORK CONFIGURATION FROM FORTRAN UNIT 01
C
C       NODE SPECIFICATIONS
C
        READ(1,*)NN
        DO I=1,NN
            READ(1,*)FRSTOU(I),LASTOU(I)
        END DO
C
C       LINK SPECIFICATIONS
C
        READ(1,*)NA
C
C       BITRATE(I) IS A PARAMETER ASSOCIATED WITH LINK I. IN THE
C       DATA NETWORK ROUTING CONTEXT IT HAS THE MEANING OF
C       TRANSMISSION CAPACITY OF LINK I.
C
        DO I=1,NA
            READ(1,*)STARTNODE(I),ENDNODE(I),BITRATE(I)
        END DO
C
C       INPUT COMMODITY DATA FROM FORTRAN UNIT 02
```

```
C
      READ(2,*)NUMCOMMOD
      DO I=1,NUMCOMMOD
         READ(2,*)ORGID(I),STARTOD(I)
      END DO
      READ(2,*)NUMODPAIR
      DO I=1,NUMODPAIR
         READ(2,*)DEST(I),INPUT_FLOW(I)
      END DO
      RETURN
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       MULTIFLO
C
C       MULTICOMMODITY FLOW ALGORITHM BASED ON A PATH FLOW FORMULATION
C       UPDATES THE PATH FLOWS OF OD PAIRS ONE AT A TIME ACCORDING TO
C       AN ITERATION OF THE PROJECTION TYPE.
C
C       DEVELOPED BY DIMITRI BERTSEKAS, BOB GENDRON, AND WEI K TSAI
C
C       BASED ON THE PAPERS:
C
C               1)  BERTSEKAS,D.P., "A CLASS OF OPTIMAL ROUTING ALGORITHMS
C                   FOR COMMUNICATION NETWORKS", PROC. OF 5TH ITERNATIONAL
C                   CONFERENCE ON COMPUTER COMMUNICATION (ICCC-80),
C                   ATLANTA, GA., OCT. 1980, PP.71-76.
C
C               2)  BERTSEKAS,D.P. AND GAFNI,E.M., "PROJECTION METHODS
C                   FOR VARIATIONAL INEQUALITIES WITH APPLICATION TO
C                   THE TRAFFIC ASSIGNMENT PROBLEM", MATH. PROGR. STUDY,17,
C                   D.C.SORENSEN AND J.-B. WETS (EDS), NORTH-HOLLAND,
C                   AMSTERDAM,1982, PP. 139-159.
C
C               3)  BERTSEKAS,D.P., "OPTIMAL ROUTING AND FLOW CONTROL
C                   METHODS FOR COMMUNICATION NETWORKS", IN ANALYSIS AND
C                   OPTIMIZATION OF SYSTEMS, (PROC. OF 5TH INTERNATIONAL
C                   CONFERENCE ON ANALYSIS AND OPTIMIZATION, VERSAILLES,
C                   FRANCE), A. BENSOUSSAN AND J.L. LIONS (EDS),
C                   SPRINGER-VERLAG, BERLIN & NY,1982, PP. 615-643.
C
C               4)  BERTSEKAS,D.P. AND GAFNI, E.M., "PROJECTED NEWTON
C                   METHODS AND OPTIMIZATION OF MULTICOMMODITY FLOWS",
C                   IEEE TRANSACTIONS ON AUTOMATIC CONTROL, DEC. 1983.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE MULTIFLO
C
        IMPLICIT NONE
C
C       ****************  INCLUDE COMMON BLOCKS  ******************************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
        INCLUDE 'PATHS.BLK'
C
C       NODE ARRAYS (LENGTH NN):
C
C       FRSTOU(NODE) - FIRST ARC OUT OF NODE
C       LASTOU(NODE) - LAST ARC OUT OF NODE
C           NOTE: THE ARC LIST MUST BE ORDERED IN SEQUENCE SO
C                 THAT ALL ARCS OUT OF ANY NODE ARE GROUPED TOGETHER
C
C       ARC ARRAYS (LENGTH NA):
C
C       FA(ARC) - THE TOTAL FLOW OF ARC
C       STARTNODE(ARC) - THE HEAD NODE OF ARC
C       ENDNODE(ARC) - THE TAIL NODE OF ARC
C
```

```
C       COMMODITY LENGTH ARRAYS (LENGTH NUMCOMMOD):
C
C       ORGID(COMMODITY) - THE NODE ID OF THE ORIGIN OF COMMODITY
C       STARTOD(COMMODITY) - THE STARTING OD PAIR IN THE ODPAIR LIST
C                       CORRESPONDING TO THE ORIGIN IN POSITION RANK
C         NOTE: THIS SCHEME ASSUMES THAT OD PAIRS ARE LISTED IN SEQUENCE
C               I.E. THE OD PAIRS CORRESPONDING TO THE COMMODITY ONE
C               ARE LISTED FIRST. THEY ARE
C               FOLLOWED BY THE OD PAIRS OF THE COMMODITY TWO
C               AND SO ON.
C
C       ODPAIR ARRAYS (LENGTH NUMOD):
C       DEST(OD) - GIVES THE DESTINATION OF ODPAIR OD
C       INPUT_FLOW(OD) - GIVES THE INPUT TRAFFIC OF ODPAIR OD
C
C       PATH ARRAYS (LENGTH DYNAMICALLY UPDATED):
C       PATHID(PATH) - THE ITERATION # AT WHICH PATH WAS GENERATED
C       NEXTPATH(PATH) - THE NEXT PATH FOR THE SAME OD PAIR FOLLOWING
C               PATH. IT EQUALS 0 IF PATH IS THE LAST FOR THAT OD PAIR
C       FP(PATH) - THE FLOW CARRIED BY PATH
C
C       PATH DESCRIPTION LIST ARRAY (LENGTH MAXITER*NUMCOMD*NN)
C       PRED(NODE,ITER,COMMODITY) - THIS TRIPLE INDEXED ARRAY SPECIFIES THE
C               SHORTEST PATH TREE GENERATED AT ITERATION ITER
C               & CORRESPONDING TO THE ORIGIN ASSOCIATED W/ COMMODITY
C               IT GIVES THE LAST ARC ON THE SHORTEST PATH FROM ORIGIN TO NODE.
C
C       ***************  LOCAL VARIABLE DEFINITIONS  ************************
C
        INTEGER*2       PRED(NNN,NMAXITER,NNORIG)
C                       PATH DESCRIPTION ARRAY - CONTAINS SHORTEST
C                       PATH TREES FOR ALL ITERATIONS
        LOGICAL SPNEW
C               LOGICAL INDICATING A NEW PATH FOUND
        LOGICAL SAME
C               LOGICAL INDICATING A NEW SHORTEST PATH ALREADY EXISTING
        INTEGER NODE
C               NODE IDENTIFIER
        INTEGER DESTOD
C               THE DESTINATION NODE OF AN OD PAIR
        INTEGER ARC
C               DO LOOP INDEX FOR ARCS
        INTEGER PATH
C               A PATH INDEX
        INTEGER NUMLIST
C               TOTAL NUMBER OF ACTIVE PATHS FOR OD PAIR UNDER CONSIDERATION
        INTEGER ITER
C               SPECIFIC ITERATION
        INTEGER N1,N2
C                TEMPORARY VARIABLES
        REAL    MINFDER
C               THE LENGTH FOR A SHORTEST PATH
        REAL    MINSDER
C               THE SECOND DERIVATIVE LENGTH FOR THE SHORTEST PATH
        REAL    TMINSDER
C               TEMPORARY VALUE FOR SECOND DERIVATIVE LENGTH OF SHORTEST PATH
        REAL    INCR
C               TOTAL SHIFT OF FLOW TO THE MINIMUM FIRST DERIVATIVE LENGTH PATH
        REAL    PATHINCR
C               SHIFT OF FLOW FOR A GIVEN PATH
```

```
      REAL    FLOW
C             FLOW FOR A PATH
      REAL    FDER
C             THE ACCRUED LENGTH ALONG A PATH
      REAL    SDER
C             THE ACCRUED SECOND DERIVATIVE LENGTH ALONG A PATH
      REAL    TEMPERROR
C             TEMPORARY STORAGE FOR CONVERGENCE ERROR
      REAL    FDLENGTH(NMAXITER)
C             ARRAY OF LENGTHS OF PATHS FOR AN OD PAIR
      REAL    SDLENGTH(NMAXITER)
C             ARRAY OF SECOND DERIVATIVE LENGTHS OF PATHS FOR AN OD PAIR
      INTEGER PATHLIST(NMAXITER)
C             ARRAY OF ACTIVE PATHS FOR AN OD PAIR
      INTEGER COMMODITY
C             DO LOOP INDEX FOR THE OD PAIR ORIGINS
      INTEGER ORIGIN
C             SPECIFIC ORIGIN
      INTEGER I
C             DO LOOP INDEX
      INTEGER OD
C             OD DO LOOP INDEX
      INTEGER K
C             DO LOOP INDEX
      INTEGER SHORTEST
C             THE SHORTEST PATH
       LOGICAL MEMBER(NNA)
C              LOGICAL FOR AN ARC INCLUDED IN THE SHORTEST PATH
      REAL    DLENGTH
C             DIFFERENCE IN PATH LENGTHS FOR THE TRAFFIC
      REAL    D1CAL
C             ARC LENGTH
      REAL    D2CAL
C             DERIVATIVE OF ARC LENGTH
C
C     **********************  EXECUTABLE CODE  ******************************
C
C     *****************************************
C     *   INITIALIZATION
C     *****************************************
C
      DO 5 ARC=1,NA
        FA(ARC)=0.0
5     CONTINUE
C
      DO I=1,NUMODPAIR
          FP(I)=INPUT_FLOW(I)
      ENDDO
      STARTOD(NUMCOMMOD+1)=NUMODPAIR+1
      NUMPATH=0
      NUMITER=1
      DO 100 COMMODITY=1,NUMCOMMOD
          ORIGIN=ORGID(COMMODITY)
          CALL SP(ORIGIN,COMMODITY)
          DO 10 I=1,NN
              PRED(I,1,COMMODITY)=PA(I)
10        CONTINUE
C
C         LOOP OVER OD PAIRS OF COMMODITY
C
```

```
            N1=STARTOD(COMMODITY)
            N2=STARTOD(COMMODITY+1)-1
             DO 50 OD=N1,N2
                 NUMPATH=NUMPATH+1
                 PATHID(NUMPATH)=1
                 NEXTPATH(NUMPATH)=0
                 FLOW=FP(NUMPATH)
                 NODE=DEST(OD)
                 DO WHILE (NODE.NE.ORIGIN)
                     ARC=PA(NODE)
                     FA(ARC)=FA(ARC)+FLOW
                     NODE=STARTNODE(ARC)
                 END DO
50           CONTINUE
100      CONTINUE
C
C        INITIALIZE THE MEMBER ARRAY
C
         DO 70 ARC=1,NA
             MEMBER(ARC)=.FALSE.
70       CONTINUE
C
C        INITIALIZE THE TOTAL DELAY
C
         CALL DELAY(DTOT(NUMITER))
C
C        OUTPUT THE CURRENT INFORMATION TO DISK
C
         CALL PRFLOW
C
C        ***********************************************
C        *  END OF INITIALIZATION
C        ***********************************************
C
C        ***** START NEW ITERATION *****
C
110      NUMITER=NUMITER+1
         CURERROR=0
C
C        **** LOOP OVER ALL COMMODITIES ****
C
         DO 1000 COMMODITY=1,NUMCOMMOD
             ORIGIN=ORGID(COMMODITY)
             CALL SP(ORIGIN,COMMODITY)
             DO 150 I=1,NN
                 PRED(I,NUMITER,COMMODITY)=PA(I)
150          CONTINUE
C
C            **** LOOP OVER OD PAIRS OF COMMODITY
C
            N1=STARTOD(COMMODITY)
            N2=STARTOD(COMMODITY+1)-1
             DO 500 OD=N1,N2
C
C            CHECK IF THERE IS ONLY ONE ACTIVE PATH AND IF SO SKIP
C            THE ITERATION
C
               IF (NEXTPATH(OD).EQ.0) THEN
                 NODE=DEST(OD)
                 DO WHILE (NODE.NE.ORIGIN)
```

```
          ARC=PA(NODE)
          IF (ARC.NE.PRED(NODE,1,COMMODITY)) GO TO 180
          NODE=STARTNODE(ARC)
        END DO
        GO TO 500
      END IF
C
180   CONTINUE
C
C      MARK THE ARCS OF THE SHORTEST PATH
C
      DESTOD=DEST(OD)
      NODE=DESTOD
      DO WHILE (NODE.NE.ORIGIN)
        ARC=PA(NODE)
        MEMBER(ARC)=.TRUE.
        NODE=STARTNODE(ARC)
      END DO
C
C        GENERATE LIST OF ACTIVE PATHS FOR OD PAIR
C
        NUMLIST=1
        PATHLIST(1)=OD
        PATH=NEXTPATH(OD)
        DO WHILE (PATH.GT.0)
            NUMLIST=NUMLIST+1
            PATHLIST(NUMLIST)=PATH
            PATH=NEXTPATH(PATH)
        END DO
C
C        DETERMINE 1ST & 2ND DERIVATIVE LENGTH OF ACTIVE PATHS
C        ALSO DETERMINE WHETHER THE CALCULATED SHORTEST PATH
C        IS ALREADY IN THE LIST
C
        SPNEW=.TRUE.
        DO 200 K=1,NUMLIST
            SAME=.TRUE.
            FDER=0
            SDER=0
            TMINSDER=0
            PATH=PATHLIST(K)
            ITER=PATHID(PATH)
            NODE=DESTOD
            DO WHILE (NODE.NE.ORIGIN)
                ARC=PRED(NODE,ITER,COMMODITY)
                CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
                FDER=FDER+D1CAL
                IF (.NOT.MEMBER(ARC)) THEN
                    SDER=SDER+D2CAL
                    SAME=.FALSE.
                ELSE
                    SDER=SDER-D2CAL
                    TMINSDER=TMINSDER+D2CAL
                END IF
                NODE=STARTNODE(ARC)
            END DO
            IF (SAME) THEN
                SPNEW=.FALSE.
                SHORTEST=PATH
                FDLENGTH(K)=FDER
```

```
                    MINFDER=FDER
                    MINSDER=TMINSDER
                 ELSE
                    FDLENGTH(K)=FDER
                    SDLENGTH(K)=SDER
                 END IF
200           CONTINUE
C
C             *** INSERT SHORTEST PATH IN PATH LIST IF IT IS NEW ***
C
              IF (SPNEW) THEN
                 NUMPATH=NUMPATH+1
                 SHORTEST=NUMPATH
                 PATHID(NUMPATH)=NUMITER
                 NEXTPATH(PATHLIST(NUMLIST))=NUMPATH
                 NEXTPATH(NUMPATH)=0
                 MINFDER=0
                 MINSDER=0
                 NODE=DESTOD
                 DO WHILE (NODE.NE.ORIGIN)
                    ARC=PA(NODE)
                    CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
                    MINFDER=MINFDER+D1CAL
                    MINSDER=MINSDER+D2CAL
                    NODE=STARTNODE(ARC)
                 END DO
              END IF
C
C             **** UPDATE PATH & LINK FLOWS ****
C
                 INCR=0
                 TEMPERROR=0
                 DO 250 K=1,NUMLIST
                    DLENGTH=FDLENGTH(K)-MINFDER
                    IF (DLENGTH.GT.0) THEN
                       PATH=PATHLIST(K)
                       FLOW=FP(PATH)
                IF ((FLOW.EQ.0.0).AND.(K.GT.1)) THEN
                  NEXTPATH(PATHLIST(K-1))=NEXTPATH(PATH)
                  GO TO 250
                END IF
                PATHINCR=DLENGTH/(SDLENGTH(K)+MINSDER)
                IF (FLOW.LE.PATHINCR) THEN
                  FP(PATH)=0.0
                  PATHINCR=FLOW
                ELSE
                  FP(PATH)=FLOW-PATHINCR
                END IF
                  INCR=INCR+PATHINCR
                  TEMPERROR=TEMPERROR+FLOW*DLENGTH/FDLENGTH(K)
                       ITER=PATHID(PATH)
                       NODE=DESTOD
                       DO WHILE (NODE.NE.ORIGIN)
                          ARC=PRED(NODE,ITER,COMMODITY)
                          FA(ARC)=FA(ARC)-PATHINCR
                          NODE=STARTNODE(ARC)
                       END DO
                    END IF
250              CONTINUE
C
```

```
C
C                    *** UPDATE THE ERROR CRITERION ***
C
                     CURERROR=AMAX1(CURERROR,TEMPERROR/INPUT_FLOW(OD))
C
C                *** UPDATE FLOWS FOR SHORTEST PATH ****
C
                 FP(SHORTEST)=FP(SHORTEST)+INCR
                 NODE=DESTOD
                 DO WHILE (NODE.NE.ORIGIN)
                     ARC=PA(NODE)
                     FA(ARC)=FA(ARC)+INCR
                     MEMBER(ARC)=.FALSE.
                     NODE=STARTNODE(ARC)
                 END DO
C
500          CONTINUE
C
C            ***** END OF LOOP FOR OD PAIRS CORRESPONDING TO COMMODITY
C            ***** UPDATE TOTAL DELAY
C
             CALL DELAY(DTOT(NUMITER))
C
1000     CONTINUE
C
C        CHECK IF THE # OF ACTIVE PATHS EXCEED THE ALLOCATED NUMBER
C
         IF (NUMPATH.GT.NNUMPATH) THEN
           WRITE(6,*)'MAX # OF ALLOCATED PATHS EXCEEDED'
           STOP
         END IF
C
C        OUTPUT THE CURRENT SOLUTION TO DISK
C
         CALL PRFLOW
C
C        ***** END OF ITERATION *****
C
C        *** IF THE ERROR IS SMALLER THAN TOL, OR THE LIMIT ON
C        THE NUMBER OF ITERATIONS IS REACHED RETURN
C        ELSE GO FOR ANOTHER ITERATION
C
         IF ((CURERROR.LT.TOL).OR.(NUMITER.EQ.MAXITER)) THEN
             RETURN
         ELSE
             GO TO 110
         END IF
C
         END
C   ************** END OF MULTIFLO ***************
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       SHORTHEAP
C       'SHORTHEAP' SOLVES THE SHORTEST PATH PROBLEM BY
C       DIJKSTRA'S ALGORITHM AND A HEAP DATA STRUCTURE.
C       THIS ALGORITHM SHOULD BE USED WHEN THE NUMBER OF
C       DESTINATIONS FOR EACH COMMODITY IS SMALL RELATIVE
C       TO THE TOTAL NUMBER OF NODES.
C
C       INPUT:
C       S - THE STARTING NODE
C       COMMODITY - THE CORRESPONDING COMMODITY
C
C       OUTPUT:
C       PA(I) - THE LAST ARC ON THE SHORTEST PATH ENDING AT NODE I
C       DIST(I) - THE SHORTEST DISTANCE TO NODE I
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE SP(S,COMMODITY)
C
        IMPLICIT NONE
C
C       *****************  INCLUDE COMMON BLOCKS  **********************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'PATHS.BLK'
C
C       ****************  LOCAL VARIABLE DEFINITIONS  ******************
C
        REAL    MIN
C               TEMPORARY MINIMUM VALUE
        REAL    D1,D2,DP
C               NODE DISTANCE
        REAL    XLARGE
C               BIG X BY DEFAULT
        INTEGER S
C               INPUT NODE
        INTEGER COMMODITY
C               INPUT COMMODITY
        INTEGER P
C               NODE ALONG THE PATH OF S TO DESTINATIONS
        INTEGER I
C               DO LOOP INDEX
        INTEGER J
C               DO LOOP INDEX
        INTEGER ARC
C               DO LOOP INDEX
        INTEGER ND
C               A NODE INDEX
        INTEGER DNUMBER
C               # OF DESTINATIONS FOR COMMODITY
        INTEGER N1
C               TEMPORARY VARIABLE
        INTEGER N2
C               TEMPORARY VARIABLE
        INTEGER UPNODE,DOWNNODE,DOWNNODE1,LASTNODE
C               VARIABLES USED IN UPDATING THE HEAP ARRAY
        INTEGER CURRANK,NEWRANK
```

```
C              VARIABLES USED IN UPDATING THE HEAP ARRAY
      INTEGER ENDHEAP
C              MARKS THE LAST ELEMENT OF THE HEAP ARRAY
      INTEGER RANK(NNN)
C              RANK(NODE) GIVES THE RANK OF NODE IN THE HEAP
      INTEGER NRANK(NNN)
C              NRANK(I) GIVES THE NODE OF RANK I IN THE HEAP
      REAL    D1CAL
C              FIRST DERIVATIVE OF DELAY WITH RESPECT TO LOAD
      LOGICAL FIRSTITER
C              TRUE IF THIS IS THE FIRST ITERATION
      LOGICAL SCAN(NNN)
C              LOGICAL INDICATING THAT A NODE HAS BEEN SCANNED
      LOGICAL DSTATUS(NNN)
C              LOGICAL SPECIFYING IF A NODE IS A DESTINATION
C
C     ******************  EXECUTABLE  CODE  ******************************
C
      XLARGE=1E15
      D1CAL=1.0
      P=S
      DO 10 I=1,NN
          DIST(I)=XLARGE
          SCAN(I)=.FALSE.
          DSTATUS(I)=.FALSE.
10    CONTINUE
      DIST(S)=0
      IF (NUMITER.EQ.1) THEN
        FIRSTITER=.TRUE.
      ELSE
        FIRSTITER=.FALSE.
      END IF
C
C     MARK THE DESTINATION NODES
C
       N1=STARTOD(COMMODITY)
       N2=STARTOD(COMMODITY+1)-1
       DNUMBER=N2-N1+1
       DO 15 I=N1,N2
         DSTATUS(DEST(I))=.TRUE.
15     CONTINUE
C
C     INITIALIZE THE HEAP FLOOR
C
      ENDHEAP=0
C
C     ***** SCAN NODE P *****
C
1000  CONTINUE
          SCAN(P)=.TRUE.
           IF (DSTATUS(P)) THEN
             IF (DNUMBER.EQ.1) RETURN
             DNUMBER=DNUMBER-1
           END IF
          IF (FRSTOU(P).NE.0) THEN
             DP=DIST(P)
              DO 20 ARC=FRSTOU(P),LASTOU(P)
                  ND=ENDNODE(ARC)
                  IF (.NOT.SCAN(ND)) THEN
                      IF (.NOT.FIRSTITER) THEN
```

```
                  CALL DERIV1(COMMODITY,FA(ARC),ARC,D1CAL)
                  END IF
                  D2=DIST(ND)
C        IF ND HAS NOT BEEN LABELLED INSERT IT IN THE HEAP
                  IF (D2.EQ.XLARGE) THEN
                    ENDHEAP=ENDHEAP+1
                    RANK(ND)=ENDHEAP
                    NRANK(ENDHEAP)=ND
                  END IF
                  D1=DP+D1CAL
                  IF (D1.LT.D2) THEN
                      PA(ND)=ARC
                      DIST(ND)=D1
                      CURRANK=RANK(ND)
50               NEWRANK=INT(CURRANK/2)
                 IF (NEWRANK.GE.1) THEN
                   UPNODE=NRANK(NEWRANK)
                   IF (D1.LT.DIST(UPNODE)) THEN
                     NRANK(CURRANK)=UPNODE
                     RANK(UPNODE)=CURRANK
                     CURRANK=NEWRANK
                     GO TO 50
                   END IF
                 END IF
                 NRANK(CURRANK)=ND
                 RANK(ND)=CURRANK
                      END IF
                  END IF
20             CONTINUE
            END IF
C
C           ******* FIND NEXT NODE TO SCAN *******
C
C          TEST FOR ERROR
            IF (ENDHEAP.EQ.0) THEN
              WRITE(6,*) 'ERROR IN THE SHORTEST PATH POUTINE'
              STOP
            END IF
            P=NRANK(1)
C
C           RESTRUCTURE HEAP ARRAYS
C
            LASTNODE=NRANK(ENDHEAP)
            ENDHEAP=ENDHEAP-1
            D1=DIST(LASTNODE)
            CURRANK=1
100         NEWRANK=CURRANK+CURRANK
            IF (NEWRANK.LE.ENDHEAP) THEN
              DOWNNODE=NRANK(NEWRANK)
                IF (NEWRANK.EQ.ENDHEAP) THEN
                  DOWNNODE1=DOWNNODE
                ELSE
                  DOWNNODE1=NRANK(NEWRANK+1)
                END IF
              IF (DIST(DOWNNODE).LE.DIST(DOWNNODE1)) THEN
                IF (D1.GT.DIST(DOWNNODE)) THEN
                  NRANK(CURRANK)=DOWNNODE
                  RANK(DOWNNODE)=CURRANK
                  CURRANK=NEWRANK
                  GO TO 100
```

```
      END IF
    ELSE
      IF (D1.GT.DIST(DOWNNODE1)) THEN
        NRANK(CURRANK)=DOWNNODE1
        RANK(DOWNNODE1)=CURRANK
        CURRANK=NEWRANK+1
        GO TO 100
      END IF
    END IF
  END IF
  NRANK(CURRANK)=LASTNODE
  RANK(LASTNODE)=CURRANK
  GO TO 1000
END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       SHORTPAPE
C       'SHORTPAPE' SOLVES THE SHORTEST PATH PROBLEM BY
C       PAPE'S MODIFICATION OF BELLMAN'S ALGORITHM.
C
C       INPUT:
C       S - THE STARTING NODE
C       COMMODITY - THE CORRESPONDING COMMODITY
C
C       OUTPUT:
C       PA(I) - THE LAST ARC ON THE SHORTEST PATH ENDING AT NODE I
C       DIST(I) - THE SHORTEST DISTANCE TO NODE I
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE SP(S,COMMODITY)
C
        IMPLICIT NONE
C
C       ******************  INCLUDE COMMON BLOCKS  ***********************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'PATHS.BLK'
C
C       *****************  LOCAL VARIABLE DEFINITIONS  *******************
C
        REAL    D1,DP
C               NODE DISTANCE
        REAL    XLARGE
C               BIG X BY DEFAULT
        INTEGER ILARGE
C               INTEGER LARGER THAN THE NUMBER OF NODES
        INTEGER S
C               INPUT NODE
        INTEGER COMMODITY
C               INPUT COMMODITY
        INTEGER P
C               NODE PRESENTLY SCANNED
        INTEGER I
C               DO LOOP INDEX
        INTEGER ARC
C               DO LOOP INDEX
        INTEGER ND
C               A NODE INDEX
        INTEGER N1
C               TEMPORARY VARIABLE
        INTEGER N2
C               TEMPORARY VARIABLE
        INTEGER ENDQUEUE
C               MARKS THE LAST ELEMENT OF THE QUEUE ARRAY
        REAL    D1CAL
C               FIRST DERIVATIVE OF DELAY WITH RESPECT TO FLOW
        LOGICAL FIRSTITER
C               TRUE IF THIS IS THE FIRST ITERATION
        INTEGER Q(NNN)
C               QUEUE OF NODES TO BE SCANNED
C
C       ******************  EXECUTABLE  CODE  ****************************
```

```
C
        XLARGE=1E15
        ILARGE=NNN+1
        D1CAL=1.0
        DO 10 I=1,NN
            DIST(I)=XLARGE
            Q(I)=0
10      CONTINUE
        IF (NUMITER.EQ.1) THEN
          FIRSTITER=.TRUE.
        ELSE
          FIRSTITER=.FALSE.
        END IF
        DIST(S)=0
        Q(S)=ILARGE
        ENDQUEUE=S
        P=S
C
C       ********* START OF MAIN ALGORITHM *********
C
100     CONTINUE
C
C       ***** SCAN NODE P *****
C
        N1=FRSTOU(P)
        IF (N1.EQ.0) GO TO 201
        N2=LASTOU(P)
        DP=DIST(P)
        DO 200 ARC=N1,N2
          ND=ENDNODE(ARC)
          IF (.NOT.FIRSTITER) THEN
             CALL DERIV1(COMMODITY,FA(ARC),ARC,D1CAL)
          END IF
          D1=DP+D1CAL
C         *** IF NO IMPROVEMENT TAKE ANOTHER ARC ***
          IF (D1.GE.DIST(ND)) GO TO 200
C         *** CHANGE DISTANCE AND LABEL OF NODE ND ***
          PA(ND)=ARC
          DIST(ND)=D1
          IF (Q(ND)) 160,140,200
C         *** IF ND HAS NEVER BEEN SCANNED INSERT IT AT THE END
C             OF THE QUEUE ***
140       Q(ENDQUEUE)=ND
          ENDQUEUE=ND
          Q(ND)=ILARGE
          GO TO 200
C         *** IF ND HAS ALREADY BEEN SCANNED ADD IT AT THE
C             BEGINNING OF THE QUEUE AFTER NODE P ***
160       Q(ND)=Q(P)
          Q(P)=ND
          IF (ENDQUEUE.EQ.P) ENDQUEUE=ND
200     CONTINUE
C
C       *** GET NEXT NODE FROM THE TOP OF THE QUEUE ***
C
201     N1=Q(P)
C
C       *** FLAG P AS HAVING BEEN SCANNED ***
C
        Q(P)=-1
```

```
      P=N1
C
C     *** IF THE QUEUE IS NOT EMPTY GO BACK TO SCAN NEXT NODE ***
C
      IF (P.LT.ILARGE) GO TO 100
C
      RETURN
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       DELAY
C
C       DELAY COMPUTES THE TOTAL M/M/1 DELAY IN ROUTING COMMODITIES FROM
C       SOURCES TO SINKS.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE DELAY(DT)
        IMPLICIT NONE
C
C       *****************  INCLUDE COMMON BLOCKS  ***************************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'PATHS.BLK'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
C
C       ********************  ARGUMENT DEFINITIONS  *************************
C
C       ON OUTPUT:
C
        REAL    DT
C               TOTAL SYSTEM DELAY
C
C       ******************  EXTERNAL FUNCTIONS REFERENCED  ******************
C
        REAL    DCAL
C               DELAY AS A FUNCTION OF FLOW
C
C       ******************  LOCAL VARIABLE DEFINITIONS  *********************
C
        INTEGER K
C               DO LOOP INDEX
C
C       ***********************  EXECUTABLE CODE  ***************************
C
C       LOOP OVER ALL LINKS AND ACCRUE TOTAL DELAY
C
        DT=0.
        DO 50 K=1,NA
            DT=DT+DCAL(FA(K),K)
 50     CONTINUE
C
        RETURN
        END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       DCAL
C
C       'DCAL' COMPUTES THE DELAY ACROSS A SPECIFIED ARC GIVEN THE FLOW.
C       THE DELAY IS ASSUMED TO BE CONSISTENT WITH M/M/1 QUEUEING FOR
C       FLOWS BELOW A MAXIMUM UTILIZATION AND QUADRATIC BEYOND WITH
C       CONTINUITY IN THE DERIVATIVES AT THE MAXIMUM UTILIZATION.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        REAL FUNCTION DCAL(X,ARC)
        IMPLICIT NONE
C
C       ******************  INCLUDE COMMON BLOCKS  ************************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
        INCLUDE 'PATHS.BLK'
C
C       *****************  ARGUMENT DEFINITIONS  **************************
C
        REAL    X
C               INPUT FLOW FOR THE ARC
        INTEGER ARC
C               INPUT ARC
C
C       ****************  LOCAL VARIABLE DEFINITIONS  *********************
C
        REAL    RATE
C               MAXIMUM LINK CAPACITY
        REAL    Y
C               TEMPORARY VARIABLE
        REAL    Z
C               TEMPORARY VARIABLE
        REAL    Q0
C               ZEROTH ORDER TERM IN THE QUADRATIC APPROXIMATION FOR
C               OVERLOADED LINKS
        REAL    Q1
C               FIRST ORDER TERM IN THE QUADRATIC APPROXIMATION
        REAL    Q2
C               SECOND ORDER TERM IN THE QUADRATIC APPROXIMATION
        REAL    EXCESS
C               FLOW BEYOND THE MAXIMUM ALLOWABLE UTILIZATION
C
C       **********************  EXECUTABLE CODE  **************************
C
        RATE=BITRATE(ARC)
        Y=MAXUTI*RATE
C
C       M/M/1 DELAY
C
        IF(X.LT.Y) THEN
            DCAL=X/(RATE-X)
        ELSE
C
C           QUADRATIC APPROXIMATION TO AVOID OVERFLOWS
C
            EXCESS=X-Y
```

```
          Z=RATE-Y
          Q0=Y/Z
          Q1=Q0/(MAXUTI*Z)
          Q2=Q1/Z
          DCAL=Q0+Q1*EXCESS+Q2*EXCESS**2
      ENDIF
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     DERIVS
C
C     'DERIVS' COMPUTES THE DERIVATIVES OF DELAY WITH RESPECT TO FLOW FOR
C     LINKS.  BELOW A MAXIMUM UTILIZATION, M/M/1 DELAY IS ASSUMED TO APPLY
C     WHEREAS A QUADRATIC APPROXIMATION IS ASSUMED FOR UTILIZATIONS BEYOND
C     THE MAXIMUM.  THE DERIVATIVES ARE CONTINUOUS AT THE MAXIMUM
C     UTILIZATION.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE DERIVS(COMMODITY,X,ARC,D1CAL,D2CAL)
      IMPLICIT NONE
C
C     ********************  INCLUDE COMMON BLOCKS  ************************
C
      INCLUDE 'PARAM.DIM'
      INCLUDE 'NETWRK.PRM'
      INCLUDE 'CONVRG.PRM'
      INCLUDE 'PATHS.BLK'
C
C     *******************  ARGUMENT DEFINITIONS  *************************
C
C     ON INPUT:
      INTEGER COMMODITY
C             THE CORRESPONDING COMMODITY
C
      REAL    X
C             FLOW IN THE SPECIFIED LINK
      INTEGER         ARC
C                     THE SPECIFIED LINK
C
C     ON OUTPUT:
C
      REAL    D1CAL
C             ARC LENGTH (1ST DERIVATIVE OF DELAY)
      REAL    D2CAL
C             FIRST DERIVATIVE OF ARC LENGTH
C
C     ****************  LOCAL VARIABLE DEFINITIONS  **********************
C
      REAL    MAXI
C             MAXIMUM ALLOWABLE FLOW FOR LINK FOR M/M/1 QUEUEING DELAY
      REAL    RATE
C             THE MAXIMUM FLOW CAPACITY FOR THE LINK
      REAL    EXCESS
C             FLOW BEYOND THE MAXIMUM ALLOWABLE FLOW
      REAL    D1
C             TEMPORARY VARIABLE
       REAL   T
C             TEMPORARY VARIABLE
```

```
C
C      *********************  EXECUTABLE CODE  ******************************
C
       RATE=BITRATE(ARC)
       MAXI=MAXUTI*RATE
       EXCESS=X-MAXI
C
       IF(EXCESS.LE.0.0) THEN
C
C          DERIVATIVES OF M/M/1 QUEUEING DELAY
C
           T=RATE-X
           D1CAL=RATE/T**2
           D2CAL=2.0*D1CAL/T
       ELSE
C
C          DERIVATIVES OF THE QUADRATIC APPROXIMATION
C
           T=RATE-MAXI
           D1=RATE/T**2
           D2CAL=2.0*D1/T
           D1CAL=D1+D2CAL*EXCESS
       END IF
       RETURN
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      DERIV1
C
C      'DERIV1' COMPUTES THE FIRST DERIVATIVE OF DELAY WITH RESPECT
C      TO FLOW FOR LINKS.  BELOW A MAXIMUM UTILIZATION, M/M/1 DELAY IS
C      ASSUMED TO APPLY WHEREAS A QUADRATIC APPROXIMATION IS ASSUMED FOR
C      UTILIZATIONS BEYOND THE MAXIMUM.  THE DERIVATIVES ARE CONTINUOUS
C      AT THE MAXIMUM UTILIZATION.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
       SUBROUTINE DERIV1(COMMODITY,X,ARC,D1CAL)
       IMPLICIT NONE
C
C      *********************  INCLUDE COMMON BLOCKS  *************************
C
       INCLUDE 'PARAM.DIM'
       INCLUDE 'NETWRK.PRM'
       INCLUDE 'CONVRG.PRM'
       INCLUDE 'PATHS.BLK'
C
C      ********************  ARGUMENT DEFINITIONS  ***************************
C
C      ON INPUT:
C
       INTEGER COMMODITY
C             THE CORRESPONDING COMMODITY
C
       REAL    X
C             FLOW IN THE SPECIFIED LINK
       INTEGER ARC
C             THE SPECIFIED ARC
C
C      ON OUTPUT:
```

```
C
      REAL     D1CAL
C              ARC LENGTH (1ST DERIVATIVE OF DELAY)
C C
C     ****************  LOCAL VARIABLE DEFINITIONS  ************************
C
      REAL     MAXI
C              MAXIMUM ALLOWABLE FLOW FOR LINK FOR M/M/1 QUEUEING DELAY
      REAL     RATE
C              THE MAXIMUM FLOW CAPACITY FOR THE LINK
      REAL     EXCESS
C              FLOW BEYOND THE MAXIMUM ALLOWABLE FLOW
      REAL     D1
C              TEMPORARY VARIABLE
      REAL     T
C              TEMPORARY VARIABLE
      REAL     D2CAL
C              TEMPORARY VARIABLE
C
C     ********************  EXECUTABLE CODE  ******************************
C
      RATE=BITRATE(ARC)
      MAXI=MAXUTI*RATE
      EXCESS=X-MAXI
C
      IF(EXCESS.LE.0.0) THEN
C
C         DERIVATIVE OF M/M/1 QUEUEING DELAY
C
          T=RATE-X
          D1CAL=RATE/T**2
      ELSE
C
C         DERIVATIVE OF THE QUADRATIC APPROXIMATION
C
          T=RATE-MAXI
          D1=RATE/T**2
          D2CAL=2.0*D1/T
          D1CAL=D1+D2CAL*EXCESS
      END IF
      RETURN
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       PRFLOW
C
C       'PRFLOW' OUTPUTS INTERMEDIATE RESULTS IN THE MULTIFLO ALGORITHM.
C       ITERATION #, DELAY, NUMBER OF ACTIVE PATHS GENERATED AND
C       CONVERGENCE ARE THE PRIMARY OUTPUTS.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE PRFLOW
        IMPLICIT NONE
C
C       ****************  INCLUDE COMMON BLOCKS  *************************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
        INCLUDE 'PATHS.BLK'
C
C       **************  LOCAL VARIABLE DEFINITIONS  ***********************
C
        LOGICAL FIRFLG
C               FIRST PASS FLAG FOR OUTPUT CONTROL
        INTEGER I
C               DO LOOP INDEX
C
C       *****************  LOCAL DATA INITIALIZATION  ******************
C
        DATA FIRFLG/.TRUE./
C
C       ON THE VERY FIRST PASS, OUTPUT THE CONTENTS OF INPUT BLOCKS TO FILE
C
        IF(FIRFLG) THEN
            WRITE(6,*)'*****************************************'
            WRITE(6,*)'*          MULTIFLO SUMMARY             *'
            WRITE(6,*)'*****************************************'
            WRITE(6,*)' '
            WRITE(6,*)'*****************************************'
            WRITE(6,*)'*          INITIALIZATION DATA          *'
            WRITE(6,*)'*****************************************'
            WRITE(6,*)' '
            WRITE(6,*)'NETWORK SPECIFICATION DATA:'
            WRITE(6,*)' '
            WRITE(6,*)'NODE SPECIFICATIONS'
            WRITE(6,*)'NUMBER OF NODES:',NN
            WRITE(6,*)'NODE #        FRSTOU        LASTOU'
            DO I=1,NN
                WRITE(6,*)I,FRSTOU(I),LASTOU(I)
            END DO
            WRITE(6,*)' '
            WRITE(6,*)'LINK SPECIFICATIONS:'
            WRITE(6,*)'NUMBER OF LINKS:',NA
            WRITE(6,*)'LINK #        STARTNODE        ENDNODE        BITRATE'
            DO I=1,NA
                WRITE(6,*)I,STARTNODE(I),ENDNODE(I),BITRATE(I)
            END DO
            WRITE(6,*)' '
            WRITE(6,*)'COMMODITY SPECIFICATIONS'
            WRITE(6,*)'NUMBER OF COMMODITIES:',NUMCOMMOD
```

```
          WRITE(6,*)'COMMOD #        ORGID           STARTOD'
          DO I=1,NUMCOMMOD
              WRITE(6,*)I,ORGID(I),STARTOD(I)
          END DO
          WRITE(6,*)' '
          WRITE(6,*)'OD PAIR SPECIFICATIONS'
          WRITE(6,*)'NUMBER OF OD PAIRS:    ',NUMODPAIR
          WRITE(6,*)'OD PAIR #       DEST            INPUT FLOW'
          DO I=1,NUMODPAIR
              WRITE(6,*)I,DEST(I),INPUT_FLOW(I)
          END DO
          WRITE(6,*)' '
          WRITE(6,*)'******************************************'
          WRITE(6,*)'*         MULTIFLO DATA BY ITERATION     *'
          WRITE(6,*)'******************************************'
          WRITE(6,*)'ITERATION #     TOTAL DELAY     CONVERGENCE  NUMBER OF'
          WRITE(6,*)'                                   ERROR        ACTIVE'
          WRITE(6,*)'                                                PATHS'
          FIRFLG=.FALSE.
      END IF
      IF(NUMITER.GT.0) THEN
          WRITE(6,*)NUMITER,DTOT(NUMITER),CURERROR,NUMPATH
      END IF
      RETURN
      END
```

```
C       'INCLUDE' FILE PARAM.DIM
C
C        'PARAM.DIM' CONTAINS THE ARRAY DIMENSIONS
C
C       *******************  NETWORK PARAMETERS  ***********************
C
        PARAMETER           NNN=100
C                           MAXIMUM NUMBER OF NODES
        PARAMETER           NNA=500
C                           MAXIMUM NUMBER OF ARCS
        PARAMETER           NNUMOD=1000
C                           MAXIMUM NUMBER OF OD PAIRS
        PARAMETER           NNUMPATH=10000
C                           MAXIMUM NUMBER OF PATHS FOR CONSIDERATION
        PARAMETER           NMAXITER=50
C                           MAXIMUM NUMBER OF ITERATIONS ALLOWED
        PARAMETER           NNORIG=100
C                           MAXIMUM NUMBER OF COMMODITIES
        PARAMETER           NINDEX=100000
C                           MAXIMUM NUMBER OF ELEMENTS OF PATH
C                           DESCRIPTION ARRAY (USED IN MULTIFLO1)
C
```

```
C         'INCLUDE' FILE NETWRK.PRM
C
C         'NETWRK.PRM' CONTAINS THE NETWORK SPECIFICATION PARAMETERS
C
          COMMON /NETWORK/
     &            NN,FRSTOU,LASTOU,
     &            NA,STARTNODE,ENDNODE,BITRATE,
     &            NUMCOMMOD,ORGID,STARTOD,
     &            NUMODPAIR,DEST,INPUT_FLOW
C
          INTEGER*2        NN
C                          NUMBER OF NODES IN THE NETWORK
          INTEGER*2        FRSTOU(NNN)
C                          THE FIRST ARC EMANATING FROM A NODE
          INTEGER*2        LASTOU(NNN)
C                          THE FINAL ARC EMANATING FROM A NODE
C
          INTEGER*2        NA
C                          NUMBER OF LINKS (ARCS) IN THE NETWORK
          INTEGER*2        STARTNODE(NNA)
C                          THE START NODE FOR AN ARC
          INTEGER*2        ENDNODE(NNA)
C                          THE END NODE FOR AN ARC
          REAL             BITRATE(NNA)
C                          THE LINK CAPACITY IN BITS/SECOND
C
          INTEGER*2        NUMCOMMOD
C                          THE NUMBER OF COMMODITIES IN THE NETWORK
          INTEGER*2        ORGID(NNORIG)
C                          THE NODE NUMBER OF THE ORIGIN
          INTEGER*2        STARTOD(NNORIG)
C                          THE POINTER TO THE STARTING NODE IN AN OD PAIR
C
          INTEGER*2        NUMODPAIR
C                          THE NUMBER OF OD PAIRS
          INTEGER*2        DEST(NNUMOD)
C                          THE DESTINATION NODE OF TRAFFIC IN AN OD PAIR
          REAL             INPUT_FLOW(NNUMOD)
C                          THE INPUT TRAFFIC TO THE NODE IN BITS/SECOND
C
```

```
C       'INCLUDE' FILE CONVRG.PRM
C
C       'CONVRG.PRM' CONTAINS THE CONVERGENCE PARAMETERS FOR THE
C       NETWORK FLOW PROBLEM
C
        COMMON /CONVRG/
     &          MAXITER,TOL,MAXUTI,OUTPFL
C
        INTEGER MAXITER
C               MAXIMUM NUMBER OF ITERATIONS IN THE SOLUTION
        REAL    TOL
C               TOLERANCE ON SOLUTION ACCURACY
        REAL    MAXUTI
C               MAXIMUM UTILIZATION FOR M/M/1 QUEUE DELAY
        LOGICAL OUTPFL
C               OUTPUT CONTROL VARIABLE
```

```
C       'INCLUDE' FILE PATHS.BLK
C
C       'PATHS.BLK' DEFINES THE ARRAYS NECESSARY TO MAINTAIN
C       PATH FLOWS AND DESCRIPTION.
C
        COMMON /PATHS/
     &          PA,FA,PATHID,NEXTPATH,FP,DIST,DTOT,CURERROR,
     &          NUMPATH,NUMITER
C
        INTEGER*2           PA(NNN)
C                           THE LAST ARC ON A SHORTEST PATH TO A NODE
        REAL                FA(NNA)
C                           THE FLOW IN ANY GIVEN LINK (ARC)
        INTEGER             PATHID(NNUMPATH)
C                           THE PATH IDENTIFIER FOR ANY GIVEN PATH
        INTEGER             NEXTPATH(NNUMPATH)
C                           THE NEXT PATH FOR THE SAME OD PAIR
        REAL                FP(NNUMPATH)
C                           THE FLOW OF A PATH
        REAL                DIST(NNN)
C                           SHORTEST DISTANCE TO A NODE FROM THE ORIGIN
        REAL                DTOT(NMAXITER)
C                           THE TOTAL DELAY BY ITERATION
        INTEGER             NUMITER
C                           CURRENT ITERATION NUMBER
        REAL                CURERROR
C                           CONVERGENCE ERROR (NORMALISED % OF FLOW NOT ON
C                           A SHORTEST PATH)
        INTEGER             NUMPATH
C                           NUMBER OF GENERATED PATHS
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       SETUP
C
C       'SETUP' ACCEPTS INPUTS FROM THE TERMINAL AND CREATES DATA SETS
C       THAT REPRESENTS NETWORKS AND LOADS IN A FORM SUITABLE FOR
C       PROGRAM 'MULTIFLO'
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        PROGRAM SETUP
        IMPLICIT NONE
C
C       ********************  INCLUDE COMMON BLOCKS  ***********************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
C
C       ***************  LOCAL VARIABLE DEFINITIONS  **********************
C
        INTEGER TERMINAL_NODE
C               THE END NODE OF A LINK
        INTEGER DESTOD
C               THE DESTINATION NODE OF AN OD PAIR
        REAL    BPS
C               MAXIMUM LINK CAPACITY
        INTEGER NUMARC
C               NUMBER OF OUTGOING ARCS FOR A NODE IN THE NETWORK
        REAL    TRAFFIC
C               SPECIFIED INPUT TO AN OD PAIR
        INTEGER I
C               DO LOOP INDEX
        INTEGER J
C               DO LOOP INDEX
        INTEGER NOD
C               NUMBER OF OD PAIRS ASSOCIATED WITH A COMMODITY
C
C       ******************  EXECUTABLE CODE  ****************************
C
C       GET THE NODE SPECIFICATIONS
C
        NA=0
        WRITE(6,*)'INPUT THE # OF NODES'
        READ(5,*)NN
        DO I=1,NN
200         WRITE(6,*)'FOR NODE',I,' ENTER # OF ARCS EXITING THE NODE'
            READ(5,*,ERR=200)NUMARC
            IF(NUMARC.GE.0) THEN
                DO J=1,NUMARC
100                 WRITE(6,*)'FOR ARC',J,' AT NODE',I,' ENTER TERMINAL NODE',
     &             ' AND MAXIMUM BITS/S'
C
C                   ASK THE SAME QUESTION ON ERRORS
C
                    READ(5,*,ERR=100)TERMINAL_NODE,BPS
                    IF(TERMINAL_NODE.GT.NN) THEN
                        WRITE(6,*)'TERMINAL NODE OUT OF BOUNDS'
                        GO TO 100
                    ELSE
C
```

```
C                       ENTER LINK BEGIN AND END NODES
C
                        NA=NA+1
                        ENDNODE(NA)=TERMINAL_NODE
                        BITRATE(NA)=BPS
                     END IF
                     STARTNODE(NA)=I
                  END DO
                  FRSTOU(I)=NA-NUMARC+1
                  LASTOU(I)=NA
               ELSE
                  WRITE(6,*)'NEGATIVE ARCS ILLEGAL'
                  GO TO 200
               END IF
            END DO
C
C           OD PAIRS SETUP
C
1000        WRITE(6,*)'ENTER THE NUMBER OF COMMODITIES IN THE NETWORK'
            READ(5,*,ERR=1000)NUMCOMMOD
            NUMODPAIR=0
            DO I=1,NUMCOMMOD
300            WRITE(6,*)'ENTER THE ORIGIN ID AND NUMBER OF DESTINATIONS FOR ',
     &            'COMMODITY',I
               READ(5,*,ERR=300)ORGID(I),NOD
               IF(ORGID(I).LE.NN) THEN
                 DO J=1,NOD
400               WRITE(6,*)'ENTER THE DESTINATION',J,' AND TRAFFIC FOR ',
     &            ' COMMODITY'
C
C                 ASK THE SAME QUESTION ON ERRORS
C
                  READ(5,*,ERR=400)DESTOD,TRAFFIC
                  IF(DESTOD.GT.NN) THEN
                     WRITE(6,*)'DESTINATION OD OUT OF BOUNDS, MAXIMUM=',NN
                     GO TO 400
                  ELSE
                     NUMODPAIR=NUMODPAIR+1
                     DEST(NUMODPAIR)=DESTOD
                     INPUT_FLOW(NUMODPAIR)=TRAFFIC
                  END IF
                 END DO
               ELSE
                  WRITE(6,*)'ORIGIN IS OUT OF BOUNDS, MAX ORIGIN=',NN
                  GO TO 300
               END IF
               STARTOD(I)=NUMODPAIR-NOD+1
            END DO
C
C           OUTPUT OF CONNECTIVITY DATA FOR DIRECT INPUT INTO 'MULTIFLO'
C           COMMON BLOCKS
C
            WRITE(1,*)NN
            DO I=1,NN
               WRITE(1,*)FRSTOU(I),LASTOU(I)
            END DO
            WRITE(1,*)NA
            DO I=1,NA
               WRITE(1,*)STARTNODE(I),ENDNODE(I),BITRATE(I)
            END DO
```

```
C
C       OUTPUT OF OD TRAFFIC DATA FOR DIRECT INPUT INTO 'MULTIFLO'
C       COMMON BLOCKS
C
        WRITE(2,*)NUMCOMMOD
        DO I=1,NUMCOMMOD
            WRITE(2,*)ORGID(I),STARTOD(I)
        END DO
        WRITE(2,*)NUMODPAIR
        DO I=1,NUMODPAIR
            WRITE(2.*)DEST(I),INPUT_FLOW(I)
        END DO
        STOP
        END
```

APPENDIX II: MULTIFLO1 Code

The only differences between MULTIFLO and MULTIFLO1 are in the DRIVER program and in the main algorithm subroutine MULTIFLO. These two routines called DRIVER1 and MULTIFLO1, are listed below.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       DRIVER1
C
C       'DRIVER1' IS A SIMPLE EXECUTIVE TO INVOKE THE 'MULTIFLO1' COMMODITY
C       ROUTING PROGRAM.  'DRIVER1' INVOKES SUBPROGRAM 'LOAD' TO READ
C       DATA INTO 'MULTIFLO1' INPUT COMMON BLOCKS.  FILES READ BY
C       'LOAD' ARE CREATED BY A TERMINAL SESSION WITH THE USER FOR
C       NETWORK DEFINITION THROUGH THE USE OF PROGRAM 'SETUP'.
C
C       EXECUTION STEPS FOR PROGRAM 'DRIVER1'
C
C               1) ASSIGN FORTRAN UNIT 01 AS CREATED BY PROGRAM 'LOAD'
C               2) ASSIGN FORTRAN UNIT 02 AS CREATED BY PROGRAM 'LOAD'
C               3) ASSIGN FORTRAN UNIT 06 AS A DESIGNATED OUTPUT FILE
C
C               E.G.:
C                   $ ASSIGN NETWORK.DAT FOR001
C                   $ ASSIGN TRAFFIC.DAT FOR002
C                   $ ASSIGN OUTPUT.DAT  FOR006
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        PROGRAM DRIVER1
C
C       LOAD FORTRAN UNIT 01 AND FORTRAN UNIT 02 FROM DISK AS CREATED
C       FROM PROGRAM 'SETUP'
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'PATHS.BLK'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
        INTEGER COMMODITY,ORIGIN,DESTOD,OD,PATH
        CALL LOAD
C
C       EXECUTE THE 'MULTIFLO1' NETWORK ALGORITHM.  'MULTIFLO1' SCHEDULES
C       ITS OWN OUTPUTS TO FORTRAN UNIT 06 ON EACH ITERATION
C
C       INITIALIZE THE TIMER
        CALL LIB$INIT_TIMER
        CALL MULTIFLO1
C       RECORD THE COMPUTATION TIME
        CALL LIB$SHOW_TIMER
C
C          PRINT MAX LINK UTILIZATION (RELEVANT FOR M/M/1 QUEUEING DELAY
C          OPTIMIZATION)
C
          UMAX=0.0
          DO 100 I=1,NA
            UMAX=MAX(UMAX,FA(I)/BITRATE(I))
100       CONTINUE
          WRITE(6,*)'MAXIMUM LINK UTILIZATION'
          WRITE(6,*)UMAX
C
C       PRINT FINAL PATH FLOW INFO
C
        WRITE(6,*)'ORIGIN / DESTINATION / PATH # / PATH FLOW'
        DO 1000 COMMODITY=1,NUMCOMMOD
          ORIGIN=ORGID(COMMODITY)
          DO 500 OD=STARTOD(COMMODITY),STARTOD(COMMODITY+1)-1
```

```
            DESTOD=DEST(OD)
            PATH=OD
            DO WHILE (PATH.GT.0)
              WRITE(6,*)ORIGIN,DESTOD,PATH,FP(PATH)
              PATH=NEXTPATH(PATH)
            END DO
500       CONTINUE
1000    CONTINUE
        STOP
        END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       MULTIFLO1
C
C       MULTICOMMODITY FLOW ALGORITHM BASED ON A PATH FLOW FORMULATION
C       UPDATES THE PATH FLOWS OF OD PAIRS ONE AT A TIME ACCORDING TO
C       AN ITERATION OF THE PROJECTION TYPE.
C
C       DEVELOPED BY DIMITRI BERTSEKAS, BOB GENDRON, AND WEI K TSAI
C
C       BASED ON THE PAPERS:
C
C               1)  BERTSEKAS,D.P., "A CLASS OF OPTIMAL ROUTING ALGORITHMS
C                   FOR COMMUNICATION NETWORKS", PROC. OF 5TH ITERNATIONAL
C                   CONFERENCE ON COMPUTER COMMUNICATION (ICCC-80),
C                   ATLANTA, GA., OCT. 1980, PP.71-76.
C
C               2)  BERTSEKAS,D.P. AND GAFNI,E.M., "PROJECTION METHODS
C                   FOR VARIATIONAL INEQUALITIES WITH APPLICATION TO
C                   THE TRAFFIC ASSIGNMENT PROBLEM", MATH. PROGR. STUDY,17,
C                   D.C.SORENSEN AND J.-B. WETS (EDS), NORTH-HOLLAND,
C                   AMSTERDAM,1982, PP. 139-159.
C
C               3)  BERTSEKAS,D.P., "OPTIMAL ROUTING AND FLOW CONTROL
C                   METHODS FOR COMMUNICATION NETWORKS", IN ANALYSIS AND
C                   OPTIMIZATION OF SYSTEMS, (PROC. OF 5TH INTERNATIONAL
C                   CONFERENCE ON ANALYSIS AND OPTIMIZATION, VERSAILLES,
C                   FRANCE), A. BENSOUSSAN AND J.L. LIONS (EDS),
C                   SPRINGER-VERLAG, BERLIN & NY,1982, PP. 615-643.
C
C               4)  BERTSEKAS,D.P. AND GAFNI, E.M., "PROJECTED NEWTON
C                   METHODS AND OPTIMIZATION OF MULTICOMMODITY FLOWS",
C                   IEEE TRANSACTIONS ON AUTOMATIC CONTROL, DEC. 1983.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE MULTIFLO1
C
        IMPLICIT NONE
C
C       ****************  INCLUDE COMMON BLOCKS  ******************************
C
        INCLUDE 'PARAM.DIM'
        INCLUDE 'NETWRK.PRM'
        INCLUDE 'CONVRG.PRM'
        INCLUDE 'PATHS.BLK'
C
C       NODE ARRAYS (LENGTH NN):
C
C       FRSTOU(NODE) - FIRST ARC OUT OF NODE
C       LASTOU(NODE) - LAST ARC OUT OF NODE
C           NOTE: THE ARC LIST MUST BE ORDERED IN SEQUENCE SO
C                 THAT ALL ARCS OUT OF ANY NODE ARE GROUPED TOGETHER
C
C       ARC ARRAYS (LENGTH NA):
C
C       FA(ARC) - THE TOTAL FLOW OF ARC
C       STARTNODE(ARC) - THE HEAD NODE OF ARC
C       ENDNODE(ARC) - THE TAIL NODE OF ARC
C
```

```
C        COMMODITY LENGTH ARRAYS (LENGTH NUMCOMMOD):
C
C        ORGID(COMMODITY) - THE NODE ID OF THE ORIGIN OF COMMODITY
C        STARTOD(COMMODITY) - THE STARTING OD PAIR IN THE ODPAIR LIST
C                    CORRESPONDING TO THE ORIGIN IN POSITION RANK
C          NOTE: THIS SCHEME ASSUMES THAT OD PAIRS ARE LISTED IN SEQUENCE
C                I.E. THE OD PAIRS CORRESPONDING TO THE COMMODITY ONE
C                ARE LISTED FIRST. THEY ARE
C                FOLLOWED BY THE OD PAIRS OF THE COMMODITY TWO
C                AND SO ON.
C
C        ODPAIR ARRAYS (LENGTH NUMOD):
C        DEST(OD) - GIVES THE DESTINATION OF ODPAIR OD
C        INPUT_FLOW(OD) - GIVES THE INPUT TRAFFIC OF ODPAIR OD
C
C        PATH ARRAYS (LENGTH DYNAMICALLY UPDATED):
C        PATHID(PATH) - POINTER TO THE BLOCK DESCRIBING PATH
C        IN THE PATH DESCRIPTION ARRAY
C        NEXTPATH(PATH) - THE NEXT PATH FOR THE SAME OD PAIR FOLLOWING
C                PATH. IT EQUALS 0 IF PATH IS THE LAST FOR THAT OD PAIR
C        FP(PATH) - THE FLOW CARRIED BY PATH
C
C        PATH DESCRIPTION LIST ARRAY (LENGTH DYNAMICALLY UPDATED)
C        PDESCR(INDEX) - THIS LONG ARRAY EXPLICITLY DESCRIBES ALL
C           ACTIVE PATHS. FOR ANY PATH, PATHID(PATH) IS A POINTER
C           TO PDESCR. IT GIVES THE ELEMENT
C           OF THE PDESCR ARRAY CONTAINING THE # OF ARCS IN THE PATH
C           (CALL IT NUMARC). THE ELEMENTS PATHID(PATH)-NUMARC TO
C           PATHID(PATH)-1 OF THE ARRAY PDESCR CONTAIN THE ARCS THAT
C           MAKE UP PATH STARTING FROM THE DESTINATION AND GOING TOWARDS
C           THE ORIGIN OF PATH.
C
C        ***************  LOCAL VARIABLE DEFINITIONS  ***********************
         INTEGER*2         PDESCR(NINDEX)
C                          PATH DESCRIPTION ARRAY - CONTAINS EXPLICIT
C                DESCRIPTION OF ALL ACTIVE PATHS.
         LOGICAL SPNEW
C                LOGICAL INDICATING A NEW PATH FOUND
         LOGICAL SAME
C                LOGICAL INDICATING A NEW SHORTEST PATH ALREADY EXISTING
         INTEGER NODE
C                NODE IDENTIFIER
         INTEGER DESTOD
C                THE DESTINATION NODE OF AN OD PAIR
         INTEGER ARC
C                DO LOOP INDEX FOR ARCS
         INTEGER PATH
C                A PATH INDEX
         INTEGER NUMLIST
C                TOTAL NUMBER OF ACTIVE PATHS FOR OD PAIR UNDER CONSIDERATION
         INTEGER ITER
C                SPECIFIC ITERATION
         INTEGER N1,N2
C                 TEMPORARY VARIABLES
         REAL    MINFDER
C                THE LENGTH FOR A SHORTEST PATH
         REAL    MINSDER
C                THE SECOND DERIVATIVE LENGTH FOR THE SHORTEST PATH
         REAL    TMINSDER
C                TEMPORARY VALUE FOR SECOND DERIVATIVE LENGTH OF SHORTEST PATH
```

```
      REAL    INCR
C             TOTAL SHIFT OF FLOW TO THE MINIMUM FIRST DERIVATIVE LENGTH PATH
      REAL    PATHINCR
C             SHIFT OF FLOW FOR A GIVEN PATH
      REAL    FLOW
C             FLOW FOR A PATH
      REAL    FDER
C             THE ACCRUED LENGTH ALONG A PATH
      REAL    SDER
C             THE ACCRUED SECOND DERIVATIVE LENGTH ALONG A PATH
      REAL    TEMPERROR
C             TEMPORARY STORAGE FOR CONVERGENCE ERROR
      REAL    FDLENGTH(NMAXITER)
C             ARRAY OF LENGTHS OF PATHS FOR AN OD PAIR
      REAL    SDLENGTH(NMAXITER)
C             ARRAY OF SECOND DERIVATIVE LENGTHS OF PATHS FOR AN OD PAIR
      INTEGER PATHLIST(NMAXITER)
C             ARRAY OF ACTIVE PATHS FOR AN OD PAIR
      INTEGER COMMODITY
C             DO LOOP INDEX FOR THE OD PAIR ORIGINS
      INTEGER ORIGIN
C             SPECIFIC ORIGIN
      INTEGER I
C             DO LOOP INDEX
      INTEGER OD
C             OD DO LOOP INDEX
      INTEGER K
C             DO LOOP INDEX
      INTEGER SHORTEST
C             THE SHORTEST PATH
      INTEGER INDEX
C             THE CURRENT LAST ELEMENT OF THE ARRAY PDESCR
      INTEGER POINT
C            POINTER TO PDESCR
      INTEGER NUMARC
C             # OF ARCS IN A PATH
      LOGICAL MEMBER(NNA)
C             LOGICAL FOR AN ARC INCLUDED IN THE SHORTEST PATH
      REAL    DLENGTH
C             DIFFERENCE IN PATH LENGTHS FOR THE TRAFFIC
      REAL    D1CAL
C             ARC LENGTH
      REAL    D2CAL
C             DERIVATIVE OF ARC LENGTH
C
C     **********************  EXECUTABLE CODE  ******************************
C
C     *****************************************
C     *   INITIALIZATION
C     *****************************************
C
      DO 5 ARC=1,NA
        FA(ARC)=0.0
5     CONTINUE
C
      DO I=1,NUMODPAIR
          FP(I)=INPUT_FLOW(I)
      ENDDO
      STARTOD(NUMCOMMOD+1)=NUMODPAIR+1
      NUMPATH=0
```

```
      INDEX=0
      NUMITER=1
      DO 100 COMMODITY=1,NUMCOMMOD
          ORIGIN=ORGID(COMMODITY)
          CALL SP(ORIGIN,COMMODITY)
C
C         LOOP OVER OD PAIRS OF COMMODITY
C
         N1=STARTOD(COMMODITY)
         N2=STARTOD(COMMODITY+1)-1
          DO 50 OD=N1,N2
              NUMPATH=NUMPATH+1
              NEXTPATH(NUMPATH)=0
              FLOW=FP(NUMPATH)
              INDEX=INDEX+1
              NUMARC=0
              NODE=DEST(OD)
              DO WHILE (NODE.NE.ORIGIN)
                  ARC=PA(NODE)
                  FA(ARC)=FA(ARC)+FLOW
                  PDESCR(INDEX)=ARC
                  NUMARC=NUMARC+1
                  INDEX=INDEX+1
                  NODE=STARTNODE(ARC)
              END DO
              PATHID(NUMPATH)=INDEX
              PDESCR(INDEX)=NUMARC
50        CONTINUE
100   CONTINUE
C
C     INITIALIZE MEMBER ARRAY
C
      DO 70 ARC=1,NA
          MEMBER(ARC)=.FALSE.
70    CONTINUE
C
C     INITIALIZE THE TOTAL DELAY
C
      CALL DELAY(DTOT(NUMITER))
C
C     OUTPUT THE CURRENT INFORMATION TO DISK
C
      CALL PRFLOW
C
C     ***********************************************
C     *  END OF INITIALIZATION
C     ***********************************************
C
C     ***** START NEW ITERATION *****
C
110   NUMITER=NUMITER+1
      CURERROR=0
C
C     **** LOOP OVER ALL COMMODITIES ****
C
      DO 1000 COMMODITY=1,NUMCOMMOD
          ORIGIN=ORGID(COMMODITY)
          CALL SP(ORIGIN,COMMODITY)
C
C          **** LOOP OVER OD PAIRS OF COMMODITY
```

```
C
            N1=STARTOD(COMMODITY)
            N2=STARTOD(COMMODITY+1)-1
             DO 500 OD=N1,N2
C
C            CHECK IF THERE IS ONLY ONE ACTIVE PATH AND IF SO SKIP
C            THE ITERATION
C
               IF (NEXTPATH(OD).EQ.0) THEN
                 NODE=DEST(OD)
                 POINT=PATHID(OD)
                 NUMARC=PDESCR(POINT)
                 DO 150 I=POINT-NUMARC,POINT-1
                   ARC=PDESCR(I)
                   IF (ARC.NE.PA(NODE)) GO TO 180
                   NODE=STARTNODE(ARC)
150              CONTINUE
                 GO TO 500
               END IF
C
180            CONTINUE
C
C               MARK THE ARCS OF THE SHORTEST PATH
C
               DESTOD=DEST(OD)
               NODE=DESTOD
               DO WHILE (NODE.NE.ORIGIN)
                 ARC=PA(NODE)
                 MEMBER(ARC)=.TRUE.
                 NODE=STARTNODE(ARC)
               END DO
C
C
C                GENERATE LIST OF ACTIVE PATHS FOR OD PAIR
C
                 NUMLIST=1
                 PATHLIST(1)=OD
                 PATH=NEXTPATH(OD)
                 DO WHILE (PATH.GT.0)
                     NUMLIST=NUMLIST+1
                     PATHLIST(NUMLIST)=PATH
                     PATH=NEXTPATH(PATH)
                 END DO
C
C                DETERMINE 1ST & 2ND DERIVATIVE LENGTH OF ACTIVE PATHS
C                ALSO DETERMINE WHETHER THE CALCULATED SHORTEST PATH
C                IS ALREADY IN THE LIST
C
                 SPNEW=.TRUE.
                 DO 200 K=1,NUMLIST
                     SAME=.TRUE.
                     FDER=0
                     SDER=0
                     TMINSDER=0
                     PATH=PATHLIST(K)
                     POINT=PATHID(PATH)
                     NUMARC=PDESCR(POINT)
                     DO 210 I=POINT-NUMARC,POINT-1
                         ARC=PDESCR(I)
                         CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
```

```
                FDER=FDER+D1CAL
                IF (.NOT.MEMBER(ARC)) THEN
                     SDER=SDER+D2CAL
                     SAME=.FALSE.
                ELSE
                     SDER=SDER-D2CAL
                     TMINSDER=TMINSDER+D2CAL
                END IF
210          CONTINUE
           IF (SAME) THEN
                SPNEW=.FALSE.
                SHORTEST=PATH
                FDLENGTH(K)=FDER
                MINFDER=FDER
                MINSDER=TMINSDER
           ELSE
                FDLENGTH(K)=FDER
                SDLENGTH(K)=SDER
           END IF
200       CONTINUE
C
C         *** INSERT SHORTEST PATH IN PATH LIST IF IT IS NEW ***
C
          IF (SPNEW) THEN
               NUMPATH=NUMPATH+1
               SHORTEST=NUMPATH
               NEXTPATH(PATHLIST(NUMLIST))=NUMPATH
               NEXTPATH(NUMPATH)=0
               MINFDER=0
               MINSDER=0
               INDEX=INDEX+1
               NUMARC=0
               NODE=DESTOD
               DO WHILE (NODE.NE.ORIGIN)
                  ARC=PA(NODE)
                  PDESCR(INDEX)=ARC
                  NUMARC=NUMARC+1
                  INDEX=INDEX+1
                  CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
                  MINFDER=MINFDER+D1CAL
                  MINSDER=MINSDER+D2CAL
                  NODE=STARTNODE(ARC)
               END DO
               PATHID(NUMPATH)=INDEX
               PDESCR(INDEX)=NUMARC
          END IF
C
C         **** UPDATE PATH & LINK FLOWS ****
C
               INCR=0
               TEMPERROR=0
               DO 250 K=1,NUMLIST
                    DLENGTH=FDLENGTH(K)-MINFDER
                    IF (DLENGTH.GT.0) THEN
                         PATH=PATHLIST(K)
                         FLOW=FP(PATH)
             IF ((FLOW.EQ.0.0).AND.(K.GT.1)) THEN
               NEXTPATH(PATHLIST(K-1))=NEXTPATH(PATH)
               GO TO 250
             END IF
```

```
                    PATHINCR=DLENGTH/(SDLENGTH(K)+MINSDER)
                    IF (FLOW.LE.PATHINCR) THEN
                      FP(PATH)=0.0
                      PATHINCR=FLOW
                    ELSE
                      FP(PATH)=FLOW-PATHINCR
                    END IF
                      INCR=INCR+PATHINCR
                      TEMPERROR=TEMPERROR+FLOW*DLENGTH/FDLENGTH(K)
                              POINT=PATHID(PATH)
                              NUMARC=PDESCR(POINT)
                              DO 220 I=POINT-NUMARC,POINT-1
                                ARC=PDESCR(I)
                                FA(ARC)=FA(ARC)-PATHINCR
220                           CONTINUE
                          END IF
250                 CONTINUE
C
C
C                   *** UPDATE THE ERROR CRITERION ***
C
                    CURERROR=AMAX1(CURERROR,TEMPERROR/INPUT_FLOW(OD))
C
C                **** UPDATE FLOWS FOR SHORTEST PATH ****
C
                 FP(SHORTEST)=FP(SHORTEST)+INCR
                 POINT=PATHID(SHORTEST)
                 NUMARC=PDESCR(POINT)
                 DO 300 I=POINT-NUMARC,POINT-1
                      ARC=PDESCR(I)
                      FA(ARC)=FA(ARC)+INCR
                      MEMBER(ARC)=.FALSE.
300              CONTINUE
C
500          CONTINUE
C
C            ***** END OF LOOP FOR OD PAIRS CORRESPONDING TO COMMODITY
C            ***** UPDATE TOTAL DELAY
C
             CALL DELAY(DTOT(NUMITER))
C
1000      CONTINUE
C
C         CHECK IF THE # OF ACTIVE PATHS EXCEED THE ALLOCATED NUMBER
C
          IF (NUMPATH.GT.NNUMPATH) THEN
            WRITE(6,*)'MAX # OF ALLOCATED PATHS EXCEEDED'
            STOP
          END IF
          IF (INDEX.GT.NINDEX) THEN
            WRITE(6,*)'DIMENSION OF PDESCR ARRAY EXCEEDED'
            STOP
          END IF
C
C         OUTPUT THE CURRENT SOLUTION TO DISK
C
          CALL PRFLOW
C
C         ***** END OF ITERATION *****
C
```

```
C       *** IF THE ERROR IS SMALLER THAN TOL, OR THE LIMIT ON
C       THE NUMBER OF ITERATIONS IS REACHED RETURN
C       ELSE GO FOR ANOTHER ITERATION
C
        IF ((CURERROR.LT.TOL).OR.(NUMITER.EQ.MAXITER)) THEN
            WRITE(6,*)'FINAL STORAGE OF PATH DESCRIPTION LIST'
            WRITE(6,*)INDEX
           RETURN
        ELSE
           GO TO 110
        END IF
C
        END
C ************** END OF MULTIFLO1 ****************
```